M

m-Estimate

►Rule Learning

Machine Learning and Game Playing

Johannes Fürnkranz
Darmstadt, Germany

Definition

Game playing is a major application area for research in artificial intelligence in general (Schaeffer & van den Herik, 2002) and for machine learning in particular (Fürnkranz & Kubat, 2001). Traditionally, the field is concerned with learning in strategy games such as tic-tac-toe (Michie, 1963), checkers (►Samuel's Checkers Player), backgammon (►TD-Gammon), chess (Baxter et al., 2000; Björnsson & Marsland, 2003; Donninger & Lorenz, 2006; Sadikov & Bratko, 2006), Go (Stern et al., 2006), Othello (Buro, 2002), poker (Billings, Peña, Schaeffer, & Szafron, 2002), or bridge (Amit & Markovitch, 2006). However, recently computer and video games have received increased attention (Laird & van Lent, 2001; Ponsen, Muñoz-Avila, Spronck, & Aha, 2006; Spronck, Ponsen, Sprinkhuizen-Kuyper, & Postma, 2006).

Motivation and Background

Since the early days of the field, game playing applications have been popular testbeds for machine learning. This has several reasons:

- *Games allow to focus on intelligent reasoning:* Other components of intelligence, such as perception or physical actions can be ignored.
- *Games are easily accessible:* A typical game-playing environment can be implemented within a few days, often hours. Exceptions are real-time computer games, for which only a few open-source test beds exist.
- *Games are very popular:* It is not very hard to describe the agent's task to the general public, and they can easily appreciate the achieved level of intelligence.

There are various types of problems that keep reoccurring in game-playing applications, for which solutions with machine learning methods are desirable, including opening book learning, learning of evaluation functions, player modeling, and others, which will be dealt with here.

Structure of the Learning System

Game-playing applications offer various challenges for machine learning. A wide variety of learning techniques have been used for tackling these problems. We cannot provide details on the learning algorithms here but will instead focus on the problems and give some of the most relevant and recent pointers to the literature. A more detailed survey can be found in Fürnkranz (2001).

Learning of Evaluation Functions

The most extensively studied learning problem in game playing is the automatic adjustment of the weights of an evaluation function. Typically, the situation is as follows: the game programmer has provided the program with a library of routines that compute important features of the current board position (e.g., the number of pieces of each kind on the board, the size of the territory controlled, etc.). What is not known is how to combine these pieces of knowledge and how to quantify their relative importance. Most frequently, these parameters are combined linearly, so that the learning task is to adjust the weights of a weighted sum. The main problem is that there are typically no direct target values that could be used as training signals. Exceptions are games

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

or endgames that have been solved completely, which are treated further below. However, in general, algorithms use ►Preference Learning (where pairs of moves or positions are labeled according to which one is preferred by an expert player) or ►Reinforcement Learning (where moves or positions are trained based on information about the eventual outcome of the game) for tuning the evaluation functions.

The key problem with reinforcement learning approaches is the ►Credit Assignment problem, i.e., even though a game has been won (lost), there might be bad (good) moves in the game. Reinforcement learning takes a radical stance at this problem, giving all positions the same reinforcement signal, hoping that erroneous signals will be evened out over time. An early classic in this area is MENACE (Michie, 1963), a tic-tac-toe player who simulates reinforcement learning with delayed rewards using a stack of matchboxes, one for each position. Each box contains a number of beads in different colors, which represent the different legal moves in the position. Moves are selected by randomly drawing a bead out of the box that represents the current position. After a game is won, all played moves are reinforced by adding beads of the corresponding colors to these boxes, and in the case of a lost game, corresponding beads are removed, thereby decreasing the probability that the same move will be played again.

The premier example of a system that has tuned its evaluation function to expert strength by playing millions of games against itself is the backgammon program ►TD-Gammon. Its key innovation was the use of a ►Neural Network instead of a position table, so that the reinforcement signal can be generalized to new unseen positions. Many authors have tried to copy TD-Gammon's learning methodology to other games (Ghory, 2004). None of these successors, however, achieved a performance that was as impressive as TD-Gammon's. The reason for this seems to be that backgammon has various characteristics that make it perfectly suited for learning from self-play. Foremost among these are the fact that the dice rolls guarantee sufficient variability, which allows to use training by self-play without the need for an explicit exploration/exploitation trade-off, and that it only requires a very limited amount of search, which allows to ignore the dependencies of search algorithm and search heuristic. These points have, for example, been addressed with limited success in the game of chess, where the program KnightCap (Baxter et al., 2000), which integrates ►Temporal Difference Learning into a game tree search by using the final positions of the principal variation for updates, and by using play on a game server for exploration.

Many aspects of evaluation function learning are still discussed in the current literature, including whether there are alternatives to reinforcement learning (e.g., evolutionary algorithms), which training strategies should be used (e.g., self-play vs. play against a teacher), etc. One of the key problems, which has already been mentioned in Samuel's Checkers Player, namely the automated construction of useful features remains largely unsolved. Some progress has, e.g., been made in the game of Othello, where a simple algorithm, very much like ►APriori has been shown to produce valuable conjunctions of basic features (Buro, 2002).

Learning Search Control

A more challenging, but considerably less investigated task is to automatically tune the various parameters that control the search in game-playing programs. These parameters influence, for example, the degree to which the search algorithm is aggressive in pruning the unpromising parts of the search tree and the lines that are explored in more depth. The key problem here is that these parameters are intertwined with the search algorithm, and cannot be optimized independently, making the process very tedious and expensive.

There have been a few attempts to use ►Explanation-Based Learning to automatically learn predicates that indicate which branches of the search tree are the most promising to follow. These approaches are quite related to various uses of ►Explanation-Based Learning in Planning, but these could not be successfully be carried over to game-tree search.

Björnsson & Marsland (2003) present a *gradient descent* approach that minimizes the total number of game positions that need to be searched in order to successfully solve a number of training problems. The idea is to adjust each parameter in proportion to its sensitivity to changes in the number of searched nodes, which is estimated with additional searches. The amount of positions that can be searched for each training position

is bounded to avoid infinite solution times for individual problems, and simulated annealing is used to ensure convergence.

Opening Book Learning

Human game players not only rely on their ability to estimate the value of moves and positions but are often also able to play certain positions "by heart," i.e., without having to think about their next move. This is the result of home preparation, opening study, and *rote learning* of important lines and variations. As computers do not forget, the use of an opening book provides an easy way for increasing their playing strength. However, the construction of such opening books can be quite laborious, and the task of keeping it up-to-date is even more challenging.

Commercial game-playing programs, in particular chess programs, have thus resorted to tools that support the automatic construction of opening from large game databases. The key challenge here is that one cannot rely on statistical information alone: a move that has been successfully employed in hundreds of games may be refuted in a single game. (Donninger & Lorenz, 2006) describe an approach that evaluates the "goodness" of a move based on a heuristic formula that has been found by experimentation. This value is then added to the result of a regular alpha-beta search. The technique has been so successful, that the chess program Hydra, probably the strongest chess program today, has abandoned conventional large man-made (and therefore error-prone) error books. Similar techniques have also been used in games like Othello (Buro, 2002).

Pattern Discovery

In addition to databases of common openings and huge game collections, which are mostly used for the tuning of evaluation functions or the automatic generation of opening books (see above), many games or subgames have already been solved, i.e., databases in which the game-theoretic value of positions of these subgames can be looked up are available. For example, all endgames with up to six pieces in chess have been solved. Other games, such as Connect-4, are solved completely, i.e., all possible positions have been evaluated and the game-theoretic value of the starting position has been determined. The largest game that has been solved so far is checkers Many of these databases are readily available, some of them (in the domains of chess, Connect-4, and tic-tac-toe) are part of the ▶UCI Repository for machine-learning databases.

The simplest learning task is to train a classifier that is able to decide whether a given game position is a game-theoretical win or loss (or draw). In many cases, this is insufficient. For example, in the chess endgame king-rook-king, any position in which the white rook cannot be immediately captured, and in which black is not stalemate is, in principle, won by white. However, in order to actually win the game it is not sufficient to simply make moves that avoid rook captures and stalemates. Thus, most databases contain the maximal number of moves that are needed for winning the position. Predicting this is a much harder, largely unsolved problem (some recent work can be found in (Sadikov & Bratko, 2006)). In addition to the game-specific knowledge that could be gained by the extraction of patterns that are indicative of won positions, another major application could be a knowledge-based compression of these databases (the collection of all perfect-play chess endgame databases with up to six men is 1.2 Terabytes in a very compressed database format, the win/loss checkers databases with up to ten men contain about 4×10^{13} positions compressed into 215GB (Schaeffer et al., 2003)).

Player Modeling

Player modeling is an important research area in game playing, which can serve several purposes. The goal of *opponent modeling* is to improve the capabilities of the machine player by allowing it to adapt to its opponent and exploit his weaknesses. Even if a game-theoretical optimal solution to a game is known, a system that has the capability to model its opponent's behavior may obtain a higher reward. Consider, for example, the game of *rock-paper-scissors* aka *RoShamBo*, in which either player can expect to win one third of the game (with one third of draws) if both players play their optimal strategies (i.e., randomly select one of their three moves). However, against a player who always plays *rock*, a player who is able to adapt his strategy to always playing *paper* can maximize his reward, while a player who sticks with the "optimal" random strategy will still win only one third of the game. One of the grand challenges in this line of work are games such as poker, in

which opponent modeling is crucial to improve over game-theoretical optimal play (Billings et al., 2002).

Player modeling is also of increasing importance in commercial computer games (see below). For one, ►Behavioral Cloning techniques could be used to increase the playing strength or credibility of artificial characters by copying the strategies of expert human players. Moreover, the playing strength of the characters can be adapted to the increasing skill level of the human player. Finally, agents that can be trained by non-programmers can also play an important role. For example, in massive multiplayer online role-playing games (MMORGs), an avatar that is trained to simulate a user's game-playing behavior could take his creator's place at times when the human player cannot attend to his game character.

Commercial Computer Games

In recent years, the computer games industry has discovered Artificial Intelligence as a necessary ingredient to make games more entertaining and challenging and, vice versa, AI has discovered computer games as an interesting and rewarding application area (Laird & van Lent, 2001). In comparison to conventional strategy games, computer game applications are more demanding, as the agents in these game typically have to interact with a large number of partner or enemy agents in a highly dynamic, real-time environment, with incomplete knowledge about its states. Tasks include off-line or on-line player modeling (see above), virtual agents with learning capabilities, optimization of plans and processes, etc.

Computer players in games are often controlled with scripts. *Dynamic scripting* (Spronck et al., 2006) is an on-line ►Reinforcement Learning technique that is designed to be integrated into scripting languages of game playing agents. Contrary to conventional reinforcement learning agents, it updates the weights of all actions for a given state simultaneously. This sacrifices guaranteed convergence, but this is desirable in a highly dynamic game environment. The approach was successfully applied to improving the strength of computer-controlled characters and increasing the entertainment value of the game by automated scaling of the difficult level of the game AI to the human player's skill level. Similar to the problem of constructing suitable features for the use in evaluation functions, the basic tactics of the computer player had to be handcoded. Ponsen et al. (2006) extend dynamic scripting with an ►Evolutionary Algorithm for automatically constructing the tactical behaviors.

Machine learning techniques are not only used for controlling players, but also for tasks like skill estimation. For example, TrueSkill^{TM} (Herbrich et al., 2007), a Bayesian skill rating system which is used for ranking players in games on the Microsoft's Xbox 360. SAGA-ML (Southey et al., 2005) is a machine learning system for supporting game designers in improving the playability of a game.

Despite the large commercial potential, research in this area has just started, and the number of workshops and publications on this topic is rapidly increasing. For more information on AI Game Development we refer to http://aigamedev.com.

Cross References

►Samuel's Checkers Player
►TD-Gammon

Recommended Reading

Amit, A., & Markovitch, S. (2006). Learning to bid in bridge. *Machine Learning, 63*(3), 287–327.

Baxter, J., Tridgell, A., & Weaver, L. (2000). Learning to play chess using temporal differences. *Machine Learning, 40*(3), 243–263.

Billings, D., Peña, L., Schaeffer, J., & Szafron, D. (2002). The challenge of poker. *Artificial Intelligence, 134*(1–2), 201–240. Special issue on games, computers and artificial intelligence.

Björnsson, Y., & Marsland, T. A. (2003). Learning extension parameters in game-tree search. *Information Sciences, 154*(3–4), 95–118.

Bowling, M., Fürnkranz, J., Graepel, T., & Musick, R. (2006). Special issue on machine learning and games. *Machine Learning, 63*(3), 211–215.

Buro, M. (2002). Improving heuristic mini-max search by supervised learning. *Artificial Intelligence, 134*(1–2), 85–99. Special issue on games, computers and artificial intelligence.

Donninger, C., & Lorenz, U. (2006). Innovative opening-book handling. In H. J. van den Herik, S.-C. Hsu, & H. H. L. M. Donkers, (Eds.), *Advances in computer games*. Berlin: Springer.

Fürnkranz, J. (2001). Machine learning in games: A survey. In J. Fürnkranz & M. Kubat (Eds.), *Machines that learn to play games* (Chap. 2, pp. 11–59). Huntington, NY: Nova Science Publishers.

Fürnkranz, J., & Kubat, M. (Eds.). (2001). *Machines that learn to play games*. Huntington, NY: Nova Science Publishers.

Ghory, I. (2004). *Reinforcement learning in board games*. Technical Report CSTR-04-004. Department of Computer Science, University of Bristol, Bristol, UK. http://www.cs.bris.ac.uk/Publications/pub_info.jsp?id=2000100.

Herbrich, R., Minka, T., & Graepel, T. (2007). Trueskill^{TM}: A bayesian skill rating system. In B. Schölkopf, J. C. Platt, &

T. Hoffman (Eds.), *Advances in neural information processing systems (NIPS-06)* (Vol. 19, pp. 569–576). Vancouver, BC, Canada: MIT Press.

Laird, J. E., & van Lent, M. (2001). Human-level AI's killler application: Interactive computer games. *AI Magazine, 22*(2), 15–26.

Michie, D. (1963). Experiments on the mechanization of game-learning – Part I. Characterization of the model and its parameters. *The Computer Journal, 6*, 232–236.

Ponsen, M., Muñoz-Avila, H., Spronck, P., & Aha, D. W. (2006). Automatically generating game tactics via evolutionary learning. *AI Magazine, 27*(3), 75–84.

Sadikov, A., & Bratko, I. (2006). Learning long-term chess strategies from databases. *Machine Learning, 63*(3), 329–340.

Schaeffer, J., Björnsson, Y., Burch, N., Lake, R., Lu, P., & Sutphen, S. (2003). Building the checkers 10-piece endgame databases. In H. J. van den Herik, H. Iida, & E. A. Heinz (Eds.), *Advances in computer games* (Vol. 10, pp. 193–210). Graz, Austria: Springer.

Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., Sutphen, S. (2007). Checkers is solved. *Science* 317(5844):1518–1522.

Schaeffer, J., & van den Herik, H. J. (Eds.) (2002). *Chips challenging champions: Games, computers and artificial intelligence*. Amsterdam: North-Holland. (Reprint of a special issue of *Artificial Intelligence, 134*(1–2)).

Southey, F., Xiao, G., Holte, R. C., Trommelen, M., & Buchanan, J. W. (2005). Semi-automated gameplay analysis by machine learning. In R. M. Young & J. E. Laird (Eds.), *Proceedings of the 1st artificial intelligence and interactive digital entertainment conference (AIIDE-05)* (pp. 123–128). AAAI Press: Marina del Rey, CA.

Spronck, P., Ponsen, M. J. V., Sprinkhuizen-Kuyper, I. G., & Postma, E. O. (2006). Adaptive game AI with dynamic scripting. *Machine Learning, 63*(3), 217–248.

Stern, D., Herbrich, R., & Graepel, T. (2006). Bayesian pattern ranking for move prediction in the game of Go. In *Proceedings of the 23rd international conference on machine learning (ICML-06)* (pp. 873–880). New York: ACM.

Machine Learning for IT Security

Philip K. Chan
Florida Institute of Technology, Melbourne, FL, USA

Definition

The prevalence of information technology (IT) across all segments of society, greatly improves the accessibility of information, however, it also provides more opportunities for individuals to act with malicious intent. Intrusion detection is the task of identifying attacks against computer systems and networks. Based on data/behavior observed in the past, machine learning methods can automate the process of building detectors for identifying malicious activities.

Motivation and Background

Cyber security often focuses on preventing attacks using authentication, filtering, and encryption techniques, but another important facet is detecting attacks once the preventive measures are breached. Consider a bank vault: thick steel doors prevent intrusions, while motion and heat sensors detect intrusions. Prevention and detection complement each other to provide a more secure environment.

How do we know if an attack has occurred or has been attempted? This requires analyzing huge volumes of data gathered from the network, host, or file systems to find suspicious activities. Two general approaches exist for this problem: *misuse detection* (also known as *signature detection*), where we look for patterns signaling well-known attacks, and *anomaly detection*, where we look for deviations from normal behavior.

Misuse detection usually works reliably on known attacks (though false alarms and missed detections are not uncommon), but has the obvious disadvantage of not being capable of detecting new attacks. Though anomaly detection can detect novel attacks, it has the drawback of not being capable of discerning intent; it can only signal that some event is unusual, but not necessarily hostile, thus generating false alarms. A desirable system would employ both approaches. Misuse detection methods are more well understood and widely applied; however, anomaly detection is much less understood and more challenging.

Can we automate the process of building software for misuse and anomaly detection? Machine learning techniques hold promise in efficiently analyzing large amounts of recent activities, identifying patterns, and building detectors.

Besides computer attacks, spam email messages, though not intended to damage computer systems or data, are annoying and waste system resources. To construct spam detectors from large amounts of email messages, machine learning techniques have been used (see References and Recommended Reading for more).

Structure of Learning System

Machine learning can be used to construct models for misuse as well as anomaly detection.

Misuse Detection

For misuse detection, the machine learning goal is to identify characteristics of known attacks. One approach is to learn the difference between attacks and normal events, which can be casted as a classification problem. Given examples of labeled attacks and normal events, a learning algorithm constructs a model that differentiates attacks from normal events.

Lee, Stolfo, and Mok (1999) apply machine learning to detect attacks in computer networks. They first identify frequent episodes, associations of features that frequently appear within a time frame, in attack and normal data separately. Frequent episodes that only appear in attack data help construct features for the models. For example, if the SYN flag is set for a http connection is a frequent episode within 2 s and the episode only appears in the attack data, a feature is constructed for the number of http connections with the SYN flag set within a period of 2 s. Using RIPPER and based on different sets of features, they construct three models: traffic, host-based traffic, and content models. The three models are then combined using meta-learning.

Ghosh and Schwartzbard (1999) use neural networks to identify attacks in operating systems. Based on system calls in the execution traces of normal and attack programs, they first identify a number of "examplar" sequences of system calls. For each system call sequence, they calculate the distance from the examplar sequences. The number of input nodes for the neural network is equal to the number of examplars and values for the input nodes are distances from those examplar sequences. The value for the output node is whether the system call sequence is from an attack or normal program.

Anomaly Detection

For anomaly detection, the machine learning goal is to characterize normal behavior. The learned models of normal behavior are then used to identify events that are anomalies, events that deviate from the models. Since anomalies are not always attacks, to reduce false alarms, the learned models usually provide a scoring mechanism to indicate the degree of anomaly.

Warrender, Forrest, and Pearlmutter (1999) identify anomalies in system calls in the operating systems. The model is a table of system call sequences from execution traces of normal programs. During detection, a sequence that is not in the table or occurs less than 0.001% in the training data is considered a mismatch. The number of mismatches within a locality frame of 20 sequences is the anomaly score.

Mahoney and Chan (2003) introduce the LERAD algorithm for learning rules that identify anomalies in network traffic. LERAD first uses a randomized algorithm to generate candidate rules that represent associations. It then finds a set of high quality rules that can succinctly cover the training data. Each rule has an associated probability of violating the rule. During detection, based on the probability, LERAD provides a score for anomalous events that do not conform to the rules in the learned model.

Misuse Detection: Schultz, Eskin, Zadok, and Stolfo (2001) with program executables, Maxion and Townsend (2002) with user commands.

Anomaly Detection: Sekar, Bendre, Dhurjati, and Bollinen (2001) with program execution, Apap, Honig, Hershkop, Eskin, and Stolfo (2002) with Windows Registry, Anderson, Lunt, Javitz, Tamaru, and Valdes (1995) with system resources, Lane and Brodley (1999) with user commands.

Spam detection: Bratko, Filipic, Cormack, Lynam, and Zupan (2006) with text, Fumera, Pillai, and Roli (2006) with text and embedded images.

Cross References

►Association
►Classification

Recommended Reading

Anderson, D., Lunt, T., Javitz, H., Tamaru, A., & Valdes, A. (1995). Detecting unusual program behavior using the statistical component of the next-generation intrusion detection expert system (NIDES). Technical Report SRI-CSL-95-06, SRI.

Apap, F., Honig, A., Hershkop, S., Eskin, E., & Stolfo, S. (2002). Detecting malicious software by monitoring anomalous windows registry accesses. In *Proceeding of fifth international symposium on recent advances in intrusion detection (RAID)*, (pp. 16–18). Zurich, Switzerland.

Bratko, A., Filipic, B., Cormack, G., Lynam, T., & Zupan, B. (2006). Spam filtering using statistical data compression models. *Journal of Machine Learning Research, 7*, 2673–2698.

Fumera, G., Pillai, I., & Roli, F. (2006). Spam filtering based on the analysis of text information embedded into images. *Journal of Machine Learning Research, 7*, 2699–2720.

Ghosh, A., & Schwartzbard, A. (1999). A study in using neural networks for anomaly and misuse detection. In *Proceeding of 8th USENIX security symposium* (pp. 141–151). Washington, DC.

Lane, T., & Brodley, C. (1999). Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security, 2*(3), 295–331.

Lee, W., Stolfo, S., & Mok, K. (1999). A data mining framework for building intrusion detection models. In *IEEE symposium on security and privacy* (pp. 120–132).

Mahoney, M., & Chan, P. (2003). Learning rules for anomaly detection of hostile network traffic. In *Proceeding of IEEE international conference data mining* (pp. 601–604). Melbourne, FL.

Maxion, R., & Townsend, T. (2002). Masquerade detection using truncated command lines. In *Proceeding of international conferernce dependable systems and networks (DSN)* (pp. 219–228). Washington, DC.

Schultz, M., Eskin, E., Zadok, E., & Stolfo, S. (2001). Data mining methods for detection of new malicious executables. In *Proceeding of IEEE symposium security and privacy* (pp. 38–49). Oakland, CA.

Sekar, R., Bendre, M., Dhurjati, D., & Bollinen, P. (2001). A fast automaton-based method for detecting anomalous program behaviors. In *Proceeding of IEEE symposium security and privacy* (pp. 144–155). Oakland, CA.

Warrender, C., Forrest, S., & Pearlmutter, B. (1999). Detecting intrusions using system calls: Alternative data models. In *IEEE symposium on security and privacy* (pp. 133–145). Los Alamitos, CA.

Manhattan Distance

Susan Craw
The Robert Gordon University, Scotland, UK

Synonyms

City Block distance; L_1-distance; 1-norm distance; Taxicab norm distance

Definition

The Manhattan distance between two points $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ in n-dimensional space is the sum of the distances in each dimension.

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} | x_i - y_i | .$$

It is called the Manhattan distance because it is the distance a car would drive in a city (e.g., Manhattan) where the buildings are laid out in square blocks and the straight streets intersect at right angles. This explains the other terms City Block and taxicab distances. The terms L_1 and 1-norm distances are the mathematical descriptions of this distance.

Cross References

►Case-Based Reasoning
►Nearest Neighbor

Margin

Definition

In a ►Support Vector Machine, a *margin* is the distance between a hyperplane and the closest example.

Cross References

►Support Vector Machines

Market Basket Analysis

►Basket Analysis

Markov Blanket

►Graphical Models

Markov Chain

►Markov Process

Markov Chain Monte Carlo

Claude Sammut
University of New South Wales
Sydney, Australia

Synonyms

MCMC

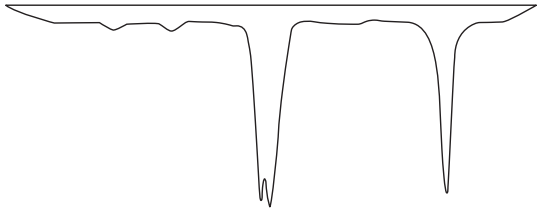
Definition

A Markov Chain Monte Carlo (MCMC) algorithm is a method for sequential sampling in which each new sample is drawn from the neighborhood of its predecessor. This sequence forms a ►Markov chain,

since the transition probabilities between sample values are only dependent on the last sample value. MCMC algorithms are well suited to sampling in high-dimensional spaces.

Motivation

Sampling from a probability density function is necessary in many kinds of approximation, including Bayesian inference and other applications in Machine Learning. However, sampling is not always easy, especially in high-dimensional spaces. Mackay (2003) gives a simple example to illustrate the problem. Suppose we want to find the average concentration of plankton in a lake, whose profile looks like this:



If we do not know the depth profile of the lake, how would we know where to sample from? If we take a boat out, would we have to sample almost exhaustively by fixing a grid on the surface of the lake and sinking our instrument progressively deeper, sampling at fixed intervals until we hit the bottom? This would be prohibitively expensive and if we had a similar problem, but with more dimensions, the problem becomes intractable. If we try to simplify the problem by drawing a random sample, how do we ensure that enough samples are taken from the canyons in the lake and not just the shallows, which account for most of the surface area?

The Algorithm

The general approach adopted in MCMC algorithms is as follows. We start sampling in some random initial state, represented by vector, $\boldsymbol{x}$. At each state, we can evaluate the probability density function, $P(\boldsymbol{x})$. We then choose a candidate next state, $\boldsymbol{x}'$, near the current state and evaluate $P(\boldsymbol{x}')$. Comparing the two, we decide whether to accept or reject the candidate. If we accept it, the candidate becomes the new current state and the process repeats for a fixed number of steps or until some convergence criterion is satisfied.

Algorithm 1 The Metropolis Algorithm

Given: target probability density function $P(\boldsymbol{x})$
a proposal distribution, Q, e.g., a Gaussian
the number of iterations, N
Output: a set of samples $\{x_i\}$ drawn from $P(\boldsymbol{x})$
Randomly select initial state vector, $\boldsymbol{x}_0$
for $i = 0$ **to** $N - 1$
  create a new candidate $\boldsymbol{x}' = \boldsymbol{x}_i + \Delta\boldsymbol{x}$,
    where $\Delta\boldsymbol{x}$ is randomly chosen from $Q(\Delta\boldsymbol{x})$
  set $\alpha = \frac{P(x')}{P(x_i)}$
  if $\alpha \geq 1$ or with probability α
    accept the new candidate and set $\boldsymbol{x}_{i+1} = \boldsymbol{x}'$
  else
    reject the candidate and set $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i$

The Metropolis Algorithm

There are several variants of the general algorithm presented above. Each variant must specify how a candidate state is proposed and what criterion should be used to accept or reject the candidate. The Metropolis algorithm assumes that the next candidate is drawn from a symmetric distribution, $Q(x)$, centered on the current state, for example, a Gaussian distribution (Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953; Metropolis & Ulam, 1949). This distribution is called the *proposal distribution*. The Metropolis algorithm is shown in Algorithm 1.

To decide if a candidate should be accepted or rejected, the algorithm calculates,

$$\alpha = \frac{P(x')}{P(x_i)}$$

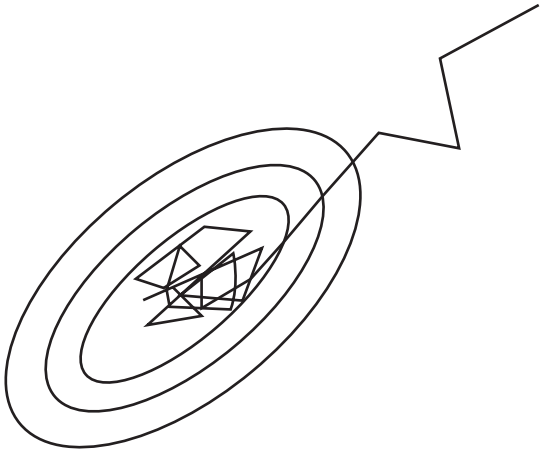
where $\boldsymbol{x}_i$ is the current state and $\boldsymbol{x}'$ is the candidate state. If $\alpha > 1$, the candidate is immediately accepted. If $\alpha < 1$, then a stochastic choice is made with the candidate being accepted with probability α, otherwise, it is rejected.

Hastings (1970) introduced a variant, the Metropolis–Hastings algorithm, which allows the proposal distribution to be asymmetric. In this case, the accept/reject calculation is:

$$\alpha = \frac{P(x')Q(x_i; x')}{P(x_i)Q(x'; x_i)}$$

Burn-in and Convergence

It can be difficult to decide how many iterations are needed before an MCMC algorithm achieves a stable distribution. Several factors affect the length of the Markov chain needed. Depending on the start state, many of the initial samples may have to be discarded, called *burn-in*, as illustrated below. The ellipses represent contours of the distribution.



The variance of the proposal distribution can also affect the chain length. If the variance is large, the jumps are large, meaning that there is varied sampling. However, this is also likely to mean that fewer samples are accepted. Narrowing the variance should increase acceptance but may require a long chain to ensure wide sampling, which is particularly necessary if the distribution has several peaks. See Andrieu et al. (2003) for a discussion of methods for improving convergence times.

Gibbs Sampling

An application of MCMC is inference in a ►Bayesian network, also known as ►Graphical Models. Here, we sample from evidence variables to find a probability for non-evidence variables. That is, we want to know what unknowns we can derive from the knowns and with what probability. Combining the evidence across a large network is intractable because we have to take into account all possible interactions of all variables, subject to the dependencies expressed in the network. Since there are too many combinations to compute in a large network, we approximate the solution by sampling.

The Gibbs sampler is a special case of the Metropolis–Hastings algorithm that is well suited to sampling from distributions over two or more dimensions. It proceeds as in Algorithm 1, except that when a new candidate is generated, only one dimension is allowed to change while all the others are held constant. Suppose we have n dimensions and $\boldsymbol{x} = (x_1, \ldots, x_n)$. One complete pass consists of jumping in one dimension, conditioned on the values for all the other dimensions, then jumping in the next dimension, and so on. That is, we initialise $\boldsymbol{x}$ to some value, and then for each x_i we resample $P(x_i|x_{j=6i})$ for j in $1 \ldots n$. The resulting candidate is immediately accepted. We then iterate, as in the usual Metropolis algorithm.

Cross References

►Bayesian Network
►Graphical Models
►Learning Graphical Models
►Markov Chain

Recommended Reading

MCMC is well covered in several text books. Mackay (2003) gives a thorough and readable introduction to MCMC and Gibbs Sampling. Russell and Norvig (2009) explain MCMC in the context of approximate inference for Bayesian networks. Hastie et al. (2009) also give a more technical account of sampling from the posterior. Andrieu et al. (2003) Machine Learning paper gives a thorough introduction to MCMC for Machine Learning. There are also some excellent tutorials on the web including Walsh (2004) and Iain Murray's video tutorial (Murray, 2009) for machine learning summer school.

Andrieu, C., DeFreitas, N., Doucet, A., & Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine Learning*, *50*(1), 5–43.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference and perception* (2nd ed.). New York: Springer.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, *57*, 97–109.

Mackay, D. J. C. (2003). *Information theory, inference and learning algorithms*. Cambridge: Cambridge University Press.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A., & Teller, H. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, *21*, 1087–1091.

Metropolis, N., & Ulam, S. (1949). The Monte Carlo method. *Journal of the American Statistical Association*, *44*(247), 335–341.

Murray, I. (2009). *Markov chain Monte Carlo*. http://videolectures.net/mlss09uk_murray_mcmc/. Retrieved 25 July 2010.

Russell, S., & Norvig, P. (2009). *Artificial intelligence: a modern approach* (3rd ed.). NJ: Prentice Hall.
Walsh, B. (2004). *Markov chain Monte Carlo and Gibbs sampling*. http://nitro.biosci.arizona.edu/courses/EEB581-2004/handouts/Gibbs.pdf. Retrieved 25 July 2010.

Markov Decision Processes

William Uther
NICTA and the University of New South Wales

Synonyms

Policy search

Definition

A *Markov Decision Process* (MDP) is a discrete, stochastic, and generally finite model of a system to which some external control can be applied. Originally developed in the Operations Research and Statistics communities, MDPs, and their extension to ▶Partially Observable Markov Decision Processes (POMDPs), are now commonly used in the study of ▶reinforcement learning in the Artificial Intelligence and Robotics communities (Bellman, 1957; Bertsekas & Tsitsiklis, 1996; Howard, 1960; Puterman, 1994). When used for reinforcement learning, firstly the parameters of an MDP are learned from data, and then the MDP is processed to choose a behavior.

Formally, an MDP is defined as a tuple: $< \mathcal{S}, \mathcal{A}, T, R >$, where $\mathcal{S}$ is a discrete set of states, $\mathcal{A}$ is a discrete set of actions, $T : \mathcal{S} \times \mathcal{A} \to (\mathcal{S} \to R)$ is a stochastic transition function, and $R : \mathcal{S} \times \mathcal{A} \to R$ specifies the expected reward received for performing the given action in each state.

An MDP carries the *Markov* label because both the transition function, T, and the reward function, R, are Markovian; i.e., they are dependent only upon the current state and action, not previous states and actions. To be a valid transition function, the distribution over the resulting states, $(\mathcal{S} \to R)$, must be a valid probability distribution, i.e., non-negative and totalling 1. Furthermore, the expected rewards must be finite.

The usual reason for specifying an MDP is to find the optimal set of actions, or *policy*, to perform. We formalize the optimality criteria below. Let us first consider how to represent a policy. In its most general form the action, $a \in \mathcal{A}$, indicated by a policy, π, might depend upon the entire history of the agent; $\pi : (\mathcal{S} \times \mathcal{A})^* \times \mathcal{S} \to \mathcal{A}$. However, for each of the common optimality criteria considered below a Markov policy, $\pi : \mathcal{S} \to \mathcal{A}$, will be sufficient. i.e., for every MDP, for each of the optimality criteria below, there exists a Markov policy that performs as well as the best full policy. Similarly, there is no requirement for an MDP that a policy be stochastic or mixed.

Optimality Criteria

Informally, one wants to choose a policy so as to maximise the long term sum of immediate rewards. Unfortunately the naive sum, $\sum_{t=0}^{\infty} r_t$ where r_t is the expected immediate reward received at time t, usually diverges. There are different optimality criteria that can than be used as alternatives.

Finite Horizon The easiest way to make sure that the sum of future expected rewards is bounded is to only consider a fixed, finite time into the future; i.e., find a policy that maximises $\sum_{t=0}^{n} r_t$ for each state.

Infinite Horizon Discounted Rather than limiting the distance we look into the future, another approach is to *discount* rewards we will receive in the future by a multiplicative factor, γ, for each time-step. This can be justified as an inflation rate, as an otherwise unmodelled probability that the simulation ends each time-step, or simply as a mathematical trick to make the criteria converge. Formally we want a policy that maximises $\sum_{t=0}^{\infty} \gamma^t r_t$ for each state.

Average Reward Unfortunately, the infinite horizon discounted optimality criterion adds another parameter to our model: the discount factor. Another approach is to optimize the average reward per time-step, or *gain*, by finding a policy that maximizes $\lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n} r_t$ for each state. This is very similar to using *sensitive discount optimality*; finding a policy that maximizes the infinite horizon discounted reward as the discount factor approaches 1, $\lim_{\gamma\to 1} \sum_{t=0}^{\infty} \gamma^t r_t$, for each state.

When maximizing average reward, any finite deviation from the optimal policy will have negligible effect on the average over an infinite timeframe. This can

make the agent "lazy." To counteract this, often a series of increasingly strict optimality criteria are used. The first is the "gain" optimality criterion given above – optimizing the long term average reward. The next is a "bias" optimality which selects from among all gain optimal policies the ones that also optimize transient initial rewards.

Value Determination

For the finite horizon, infinite horizon discounted, or bias optimality criteria, the optimality criteria can be calculated for each state, or for each state-action pair, giving a *value function*. Once found, the value function can then be used to find an optimal policy.

Bellman Equations The standard approach to finding the value function for a policy over an MDP is a dynamic programming approach using a recursive formulation of the optimality criteria. That recursive formulation is known as the Bellman Equation.

There are two, closely related, common forms for a value function; the state value function, $V : \mathcal{S} \to R$ and the state-action value function, $Q : \mathcal{S} \times \mathcal{A} \to R$. For a finite horizon undiscounted optimality criterion with time horizon n and policy π:

$$\begin{aligned} Q_n^\pi(s,a) &= \sum_{t=0}^{n} r_t \\ &= R(s,a) + \mathrm{E}_{s' \in T(s,a)} V_{n-1}^\pi(s') \\ &= R(s,a) + \sum_{s' \in \mathcal{S}} T(s,a)(s') V_{n-1}^\pi(s') \\ V_n^\pi(s) &= Q_n^\pi(s, \pi(s)) \end{aligned}$$

For the infinite horizon discounted case:

$$\begin{aligned} Q^\pi(s,a) &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') V^\pi(s') \\ V^\pi(s) &= Q^\pi(s, \pi(s)) \end{aligned}$$

These equations can be turned into a method for finding the value function by replacing the equality with an assignment:

$$Q^\pi(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') Q^\pi(s', \pi(s'))$$

As long as this update rule is followed infinitely often for each state/action pair, the Q-function will converge.

Prioritised sweeping: Rather than blindly updating each state/action, intelligent choice of where to update will significantly speed convergence. One technique for this is called Prioritized Sweeping (Andre et al., 1997; Moore & Atkeson, 1993).

A priority queue of states is kept. Initially one complete pass of updates over all states is performed, but thereafter states are updated in the order they are pulled from the priority queue. Any time the value of a state, $V^\pi(s)$, changes, the priorities of all states, s', that can reach state s are updated; we update $\{s' \mid T(s', \pi(s'))(s) \neq 0\}$. The priorities are increased by the absolute change in $V^\pi(s)$.

The effect of the priority queue is to focus computation where values are changing rapidly.

Linear Programming Solutions Rather than using the Bellman equation and dynamic programming, an alternative approach is to set up a collection of inequalities and use linear programming to find an optimal value function. In particular if we minimize,

$$\sum_{s \in \mathcal{S}} V^\pi(s)$$

subject to the constraints

$$\forall_{s \in \mathcal{S}}\ 0 \leq V^\pi(s) - \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') V^\pi(s') \right],$$

then the resulting V^π accurately estimates the expected sum of discounted reward.

Bellman Error Minimization A third approach to value determination is similar to the dynamic programming solution above. Rather than replacing the equality in the Bellman equation with an assignment, it turns the equation into an error function and adjusts the Q function to minimise the sum of squared Bellman residuals (Baird, 1995):

$$\mathrm{Residual}(s) = Q^\pi(s,a) - \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') Q^\pi(s', \pi(s')) \right] \mathrm{Err} = \sum_{s \in \mathcal{S}} \mathrm{Residual}(s)^2$$

Control Methods

The previous section gave us a way to obtain a value function for a particular policy, but what we usually need is a good policy, not a value function for the policy we already have. For an optimal policy, for each state:

$$\pi(s) = argmax_{a \in \mathcal{A}} Q^{\pi}(s,a)$$

If a policy, π, is not optimal then its value function can be used to find a better policy, π'. It is common to use the greedy policy for the value function:

$$\pi'(s) \leftarrow argmax_{a \in \mathcal{A}} Q^{\pi}(s,a)$$

This process can be used iteratively to find the optimal policy.

Policy iteration: Policy iteration alternates between value determination and greedy policy updating steps until convergence is achieved. The algorithm starts with a policy, π_1. The value function is calculated for that policy, V^{π_1}. A new policy is then found from that value function, π_2. This alternation between finding the optimal value function for a given policy and then improving the policy continues until convergence. At convergence the policy is optimal.

Value iteration: Rather than explicitly updating the policy, value iteration works directly with the value function. We define an update,

$$Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') \max_{a' \in \mathcal{A}} Q(s',a'),$$

with a maximization step included. As long as this update is performed often enough in each state, Q will converge. Once Q has converged, the greedy policy will be optimal.

Mixed policy iteration: The two previous methods, policy and value iteration, are two extremes of a spectrum. In practice updates to the policy and value function can occur asynchronously as long as the value and policy in each state are updated often enough.

Representations

In the above discussion we have discussed a number of functions, but not discussed how these functions are represented. The default representation is an array or tabular form which has no constraints on the function it can represent. However, the ▶curse of dimensionality suggests that the number of states will, in general, be exponential in the problem size. This can make even a single complete iteration over the state space intractable. One solution is to represent the functions in a more compact form so that they can be updated efficiently. This approach is known as *function approximation*. Here we review some common techniques.

A class of representations is chosen to represent the functions we need to process: e.g., the transition, T, reward, R, Value, V or Q, and/or policy, π, functions. A particular function is selected from the chosen class by a parameter vector, $\boldsymbol{\theta}$.

There are two important questions that must be answered by any scheme using function approximation; does the resulting algorithm converge to a solution, and does the resulting solution bear any useful relationship with the optimal solution?

A simple approach when using a differentiable function to represent the value function is to use a form of ▶temporal difference learning. For a given state, s, and action, a, the Bellman equation is used to calculate a new value, $Q^{\text{new}}(s,a)$, and then $\boldsymbol{\theta}$ is updated to move the value function toward this new value. This gradient based approach usually has a learning rate, $\alpha \in [0,1]$, to adjust the speed of learning.

$$Q^{\text{new}}(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a)(s') V^{\text{old}}(s')$$

$$\boldsymbol{\Delta}_{s,a}\boldsymbol{\theta} = \alpha \frac{\partial Q}{\partial \boldsymbol{\theta}} (Q^{\text{new}}(s,a) - Q^{\text{old}}(s,a))$$

This approach is known not to converge in general, although it does converge in some special cases. A similar approach with full Bellman Error minimization will not oscillate, but it may cause the $\boldsymbol{\theta}$ to diverge even as the Bellman residual converges.

Contraction mappings: The first class of function approximators that was shown to converge with the above update, apart from a complete tabular representation, was the class of contraction mappings (Gordon, 1995). Simply put, these are function approximation classes where changing one value by a certain amount changes every other value in the approximator by no more than that amount. For example, linear interpolation and *tile coding* (Tile codings are also known as Cerebellar Motor Action Controllers (CMAC) in early work (Albus, 1981)) are each contraction mappings whereas linear extrapolation is not.

Formally, let $\mathcal{S}$ be a vector space with max norm $||.||_\infty$. A function f is a contraction mapping if,

$$\forall a, b \in \mathcal{S}, ||f(a) - f(b)||_\infty < ||a - b||_\infty$$

The class of function approximations that form contraction mappings includes a number of common approximation techniques including *tile coding*. Tile coding represents a function as a linear combination of basis functions, $\boldsymbol{\varphi}(s, a)$,

$$\hat{Q}(s, a) = \boldsymbol{\theta} \,.\, \varphi(s, a),$$

where the individual elements of φ are binary features on the underlying state.

Linear approximations: The linear combination of basis functions can be extended beyond binary features. This will converge when temporal differencing updates are performed in trajectories through the state space following the policy being evaluated (Tsitsiklis & Van Roy, 1997).

Variable resolution techniques: One technique for representing value functions over large state spaces is use a non-parametric representation. Munos gives a technique that introduces more basis functions for their approximation over time as needed (Munos & Moore, 2001).

Dynamic Bayesian networks: ►Bayesian Networks are an efficient representation of a factored probability distribution. Dynamic Bayesian Networks use the Bayesian Network formalism to represent the transition function, $\mathcal{T}$, in an MDP (Guestrin et al., 2003). The reward and value functions are usually represented with linear approximations. The policy is usually represented implicitly by the value function.

Decision diagrams: Arithmetic Decision Diagrams (ADDs) are a compact way of representing functions from a factored discrete domain to a real range. ADDs can also be efficiently manipulated, with operators for the addition and multiplication of ADDs as well as taking the maximum of two ADDs. As the Bellman equation can be re-written using operators, it is possible to implement mixed policy iteration using this efficient representation (St-Aubin et al., 2000).

Hierarchical representations: ►Hierarchical Reinforcement Learning factors out common substructure in the functions that represent an MDP in order to solve it efficiently. This has been done in many different ways. Dietterich's *MAXQ* hierarchy allowed a prespecified hierarchy to re-use common elements in a value function (Dietterich, 2000). Sutton's *Options* framework focussed on temporal abstraction and re-use of policy elements (Sutton et al., 1998). Moore's *Airports* hierarchy allowed automatic decomposition of a problem where the specific goal could change over time, and so was made part of the state (Moore et al., 1999). Andre's *A-Lisp* system takes the hierarchical representation to an extreme by building in a Turing complete programming language (Andre & Russell, 2002).

Greedy Algorithms Versus Search

In the previous sections the control problem was solved using a greedy policy for a value function. If the value function was approximate, then the resulting policy may be less than optimal. Another approach to improving the policy is to introduce search during execution. Given the current state, the agent conducts a forward search looking for the sequence of actions that produces the best intermediate reward and resulting state value combination.

These searches can be divided into two broad categories: deterministic and stochastic searches. Deterministic searches, such as LAO* (Hansen & Zilberstein, 1998), expand through the state space using the supplied model of the MDP. In contrast stochastic, or Monte-Carlo, approaches sample trajectories from the model and use statistics gathered from those samples to choose a policy (Kocsis & Szepesvári, 2006).

Cross References

►Bayesian Network
►Curse of Dimensionality
►Monte-Carlo Simulation
►Partially Observable Markov Decision Processes
►Reinforcement Learning
►Temporal Difference Learning

Recommended Reading

Albus, J. S. (1981). *Brains, behavior, and robotics*. Peterborough: BYTE. ISBN: 0070009759.

Andre, D., Friedman, N., & Parr, R. (1997). Generalized prioritized sweeping. *Neural and Information Processing Systems*, pp. 1001–1007.

Andre, D., Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*.

Baird, L. C. (1995). Residual algorithms: reinforcement learning with function approximation. In A. Prieditis & S. Russell (Eds.), *Machine Learning: Proceedings of the Twelfth International Conference (ICML95)* (pp. 30–37). San Mateo: Morgan Kaufmann.

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

Bertsekas, D. P., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research 13*, 227–303.

Gordon, G. J. (1995). *Stable function approximation in dynamic programming* (Technical report CMU-CS-95-103). School of Computer Science, Carnegie Mellon University.

Guestrin, C., et al. (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research, 19*, 399–468.

Hansen, E. A., & Zilberstein, S. (1998). Heuristic search in cyclic AND/OR graphs. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. http://rbr.cs.umass.edu/shlomo/papers/HZaaai98.html

Howard, R. A. (1960). *Dynamic programming and Markov processes*. Cambridge: MIT Press.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. *European Conference on Machine Learning (ECML). Lecture Notes in Computer Science 4212*, Springer, pp. 282–293.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: reinforcement learning with less data and less real time. *Machine Learning, 13*, 103–130.

Moore, A. W., Baird, L., & Pack Kaelbling, L. (1999). Multi-value-functions: efficient automatic action hierarchies for multiple goal MDPs. *International Joint Conference on Artificial Intelligence (IJCAI99)*.

Munos, R., & Moore, A. W. (2001). Variable resolution discretization in optimal control. *Machine Learning, 1*, 1–31.

Puterman, M. L. (1994). Markov decision processes: discrete stochastic dynamic programming. Wiley series in probability and mathematical statistics. Applied probability and statistics section. New York: Wiley. ISBN: 0-471-61977-9.

St-Aubin, R., Hoey, J., & Boutilier, C. (2000). APRICODD: approximate policy construction using decision diagrams. NIPS-2000.

Sutton, R. S., Precup, D., & Singh, S. (1998). Intra-option learning about temporally abstract actions. *Machine Learning: Proceedings of the Fifteenth International Conference (ICML98)*, Morgan Kaufmann, Madison, pp. 556–564.

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control, 42*(5), 674–690.

Markov Model

►Markov Process

Markov Net

►Markov Network

Markov Network

Synonyms

Markov net; Markov random field

Definition

A Markov network is a form of undirected ►graphical model for representing multivariate probability distributions.

Cross References

►Graphical Models

Markov Process

Synonyms

Markov chain; Markov model

A stochastic process in which the conditional probability distribution of future states of the process, given the present state and all past states, depends only upon the present state. A process with this property may be called Markovian. The best known Markovian processes are *Markov chains*, also known as *Markov Models*, which are discrete-time series of states with transition probabilities. Markov chains are named after Andrey Markov (1865–1922), who introduced several significant new notions to the concept of stochastic processes. Brownian motion is another well-known phenomenon that, to close approximation, is a Markov process.

Recommended Reading

Meyn, S. P., & Tweedie, R. L. (1993). *Markov chains and stochastic stability*. Springer-Verlag, London

Markov Random Field

►Markov Network

Markovian Decision Rule

Synonyms

Randomized decision rule

Definition

In a ►Markov decision process, a *decision rule*, d_t, determines what *action* to take, based on the history to date at a given *decision epoch* and for any possible *state*. It is *deterministic* if it selects a single member of $A(s)$ with probability 1 for each $s \in S$ and for a given h_t, and it is *randomized* if it selects a member of $A(s)$ at random with probability $q_{d_t(h_t)}(a)$. It is *Markovian* if it depends on h_t only through s_t. That is, $d_t(h_t) = d_t(s_t)$.

Maxent Models

►Maximum Entropy Models for Natural Language Processing

Maximum Entropy Models for Natural Language Processing

Adwait Ratnaparkhi
Yahoo! Labs, Santa Clara
California, USA

Synonyms

Log-linear models; Maxent models; Statistical natural language processing

Definition

The term maximum entropy refers to an optimization framework in which the goal is to find the probability model that maximizes entropy over the set of models that are consistent with the observed evidence.

The information-theoretic notion of entropy is a way to quantify the uncertainty of a probability model; higher entropy corresponds to more uncertainty in the probability distribution. The rationale for choosing the maximum entropy model – from the set of models that meet the evidence – is that any other model assumes evidence that has not been observed (Jaynes, 1957).

In most natural language processing problems, observed evidence takes the form of co-occurrence counts between some prediction of interest and some linguistic context of interest. These counts are derived from a large number of linguistically annotated examples, known as a corpus. For example, the frequency in a large corpus with which the word *that* co-occurs with the tag corresponding to determiner, or *DET*, is a piece of observed evidence. A probability model is consistent with the observed evidence if its calculated estimates of the co-occurrence counts agree with the observed counts in the corpus.

The goal of the maximum entropy framework is to find a model that is consistent with the co-occurrence counts, but is otherwise maximally uncertain. It provides a way to combine many pieces of evidence into a single probability model. An iterative parameter estimation procedure is usually necessary to find the maximum entropy probability model.

Motivation and Background

The early 1990s saw a resurgence in the use of statistical methods for natural language processing (Church & Mercer, 1993). In particular, the IBM TJ Watson Research Center was a prominent advocate in this field for statistical methods such as the maximum entropy framework. Language modeling for speech recognition (Lau, Rosenfeld, & Roukos, 1993) and machine translation (Berger, Della Pietra, & Della Pietra, 1996) were among the early applications of this framework.

Structure of Learning System

The goal of a typical natural language processing application is to automatically produce linguistically motivated categories or structures over freely occurring text. In statistically based approaches, it is convenient to produce the categories with a conditional probability model p such that $p(a|b)$ is the probability of seeing a prediction of interest a (e.g., a part-of-speech tag) given a linguistic context of interest b (e.g., a word).

The maximum entropy framework discussed here follows the machine learning approach to NLP, which assumes the existence of a large corpus of linguistically annotated examples. This annotated corpus is used to create a training set, which in turn is used to estimate the probability model p.

Representing Evidence

Evidence for the maximum entropy model is derived from the training set. The training set is a list of (prediction, linguistic context) pairs that are generated from the annotated data. However, in practice, we do not record the entire linguistic context. Instead, linguistically motivated Boolean-valued questions reduce the entire linguistic context to a vector of question identifiers. Therefore, each training sample looks like:

Prediction	Question vector
a	$q_1 \ldots q_n$

where a is the prediction and where $q_1 \ldots q_n$ is a vector of questions that answered true for the linguistic context corresponding to this training sample. The questions must be designed by the experimenter in advance, and are specifically designed for the annotated data and the problem space.

In the framework discussed here, any piece of evidence is represented with a *feature*. A feature f_j correlates a prediction a with an aspect of a linguistic context b, captured by some question:

$$f_j(a,b) = \begin{cases} 1 & \text{if } a = x \text{ and } q(b) = \text{true} \\ 0 & \text{otherwise} \end{cases}$$

Combining the Evidence

The maximum entropy framework provides a way to combine all the features into a probability model. In the conditional maximum entropy formulation (Berger et al., 1996), the desired model p^* is given by:

$$P = \{p | E_p f_j = E_{\tilde{p}} f_j, j = 1 \ldots k\}, \quad (1)$$
$$H(p) = -\sum_{a,b} \tilde{p}(b) p(a|b) \log p(a|b),$$
$$p^* = argmax_{p \in P} H(p),$$

where $H(p)$ is the conditional entropy of p, $\tilde{p}(b)$ is the observed probability of the linguistic context b in the training set, and P is the set of models that are consistent with the observed data. A model p is consistent if its own feature expectation $E_p f_j$ is equal to the observed feature expectation $E_{\tilde{p}} f_j$, for all $j = 1 \ldots k$ features. $E_{\tilde{p}} f_j$ can be interpreted as the observed count of f_j in the training sample, normalized by the training sample size. Both are defined as follows:

$$E_p f_j = \sum_{a,b} \tilde{p}(b) p(a|b) f_j(a,b),$$
$$E_{\tilde{p}} f_j = \sum_{a,b} \tilde{p}(a,b) f_j(a,b).$$

According to the maximum entropy framework, the optimal model p^* is the most uncertain model among those that satisfy the feature constraints. It is possible to show that the form of the optimal model must be log-linear:

$$p^*(a|b) = \frac{1}{Z(b)} \prod_{j=1 \ldots k} \alpha_j^{f_j(a,b)}, \quad (2)$$

$$Z(b) = \sum_{a'} \prod_{j=1 \ldots k} \alpha_j^{f_j(a',b)}.$$

Here $Z(b)$ is a normalization factor, and $\alpha_j > 0$. Each model parameter α_j can be viewed as the "strength" of its corresponding feature f_j; the conditional probability is the normalized product of the feature weights of the active features.

Relationship to Maximum Likelihood

The maximum entropy framework described here has an alternate interpretation under the more commonly used technique of maximum likelihood estimation.

$$Q = \left\{ p | p(a|b) = \frac{1}{Z(b)} \prod_{j=1 \ldots k} \alpha_j^{f_j(a,b)} \right\},$$
$$L(p) = \sum_{a,b} \tilde{p}(a,b) \log p(a|b),$$
$$q^* = argmax_{p \in Q} L(p).$$

Here Q is the set of models of form (2), $\tilde{p}(a,b)$ is the observed probability of prediction a together with linguistic context b, $L(p)$ is the log-likelihood of the training set, and q^* is the maximum likelihood model. It can be shown that $p^* = q^*$; maximum likelihood estimation for models of the form (2) gives the same answer as maximum entropy estimation over the constraints on feature counts (1). The difference between

approaches is that the maximum likelihood approach assumes the form of the model, whereas the maximum entropy approach assumes the constraints on feature expectations, and *derives* the models form.

Parameter Estimation

The Generalized Iterative Scaling (GIS) algorithm (Darroch & Ratcliff, 1972) is the easiest way to estimate the parameters for this kind of model. The iterative updates are given below:

$$\alpha_j^{(0)} = 1,$$

$$\alpha_j^{(n)} = \alpha_j^{(n-1)} \left[\frac{E_{\tilde{p}} f_j}{E_p f_j} \right]^{\frac{1}{C}}.$$

GIS requires the use of a "correction" feature g and constant $C > 0$, which are defined so that $g(a,b) = C - \sum_{j=1\ldots k} f_j(a,b)$ for any (a,b) pair in the training set. Normally, the correction feature g must be trained in the model along with the k original features, although Curran and Clark (2003) show that GIS can converge even without the correction feature. The number of iterations needed to achieve convergence depends on certain aspects of the data, such as the training sample size and the feature set size, and is typically tuned for the problem at hand.

Other algorithms for parameter estimation include the Improved Iterative Scaling (Berger et al., 1996) algorithm and the Sequential Conditional GIS (Goodman, 2002) algorithm. The list given here is not complete; many other numerical algorithms can be applied to maximum entropy parameter estimation, see (Malouf, 2002) for a comparison.

It is usually difficult to assess the reliability of features that occur infrequently in the training set, especially those that occur only once. When the parameters are trained from low frequency feature counts, maximum entropy models – as well as many other statistical learning techniques – have a tendency to "overfit" the training data. In this case, performance on training data appears very high, but performance on the intended test data usually suffers. *Smoothing* algorithms are designed to alleviate this problem for statistical models; some smoothing techniques for maximum entropy models are reviewed in (Chen & Rosenfeld, 1999).

Applications

This framework has been used as a generic machine learning toolkit for many problems in natural language processing. Like other generic machine learning techniques, the core of the maximum entropy framework is invariant across different problem spaces. However, some information is specific to each problem space:

1. Predictions: The space of predictions for this model.
2. Questions: The space of questions for this model.
3. Feature Selection: Any possible (question, prediction) pair can be used as a feature. In complex models, only a small subset of all the possible features are used in a model. The feature selection strategy specifies how to choose the subset.

For a given application, it suffices to give the above three pieces of information to fully specify a maximum entropy probability model.

Part-of-Speech Tagging

Part-of-speech tagging is a well-known task in computational linguistics in which the goal is to disambiguate the part-of-speech of all the words in a given sentence. For example, it can be non trivial for a computer to disambiguate the part-of-speech of the word *flies* in the following famous examples:

- Fruit *flies* like a banana
- Time *flies* like an arrow

The word *flies* behaves like a noun in the first case, and like a *verb* in the second case. In the machine learning approach to this problem, co-occurrence statistics of tags and words in the linguistic context are used to create a predictive model for part-of-speech tags.

The computational linguistics community has created annotated corpora to help build and test algorithms for tagging. One such corpus, known as the Penn treebank (Marcus, Santorini, & Marcinkiewicz, 1994), has been used extensively by machine learning and statistical NLP practitioners for problems like tagging. In this corpus, roughly 1M words from the Wall Street Journal have manually been assigned part-of-speech tags. This corpus can be converted into a set of training samples, which in turn can be used to train a maximum entropy model.

Model Specification For tagging, the goal is a maximum entropy model p that will produce a probability of seeing a tag at position i, given the linguistic context of

the ith word, the surrounding words, and the previously predicted tags, written as $p(t_i|t_{i-1}\ldots t_1, w_1\ldots w_n)$. The intent is to use the model left-to-right, one word at a time. The maximum entropy model for tagging (Ratnaparkhi, 1996) is specified as:

1. Predictions: The 45 part-of-speech tags of the Penn treebank
2. Questions: Listed below are the questions and question patterns. A question pattern has a placeholder variable (e.g., X, Y) that is instantiated by scanning the annotated corpus for examples in which the patterns match. Let i denote the position of the current word in the sentence, and let w_i and t_i denote the word and tag at position i, respectively.
   - Does $w_i = X$?
   - Does $w_{i-1} = X$?
   - Does $w_{i-2} = X$?
   - Does $w_{i+1} = X$?
   - Does $w_{i+2} = X$?
   - Does $t_{i-1} = X$?
   - Does $t_{i-1}t_{i-2} = X, Y$?
   - For words that occur fewer than 5 times in the training set:
     - Are the first K (for $K \le 4$) characters $X_1 \ldots X_K$?
     - Are the last K (for $K \le 4$) characters $X_1 \ldots X_K$?
     - Does the current word contain a number?
     - Does the current word contain a hyphen?
     - Does the current word contain an uppercase character?
3. Feature Selection: Any (question, prediction) pair whose count in the training data is ≥10 is retained as a feature.

While the features for each probability decision could in theory look at the entire linguistic context, they actually only look at a small window of words surrounding the current word, and a small window of tags to the left. Therefore each decision effectively makes the Markov-like assumption given in (3).

$$p(t_i|t_{i-1}\ldots t_1, w_1\ldots w_n) = p(t_i|t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_i w_{i+1}w_{i+2}) \quad (3)$$

$$= \frac{\prod_{j=1\ldots k}\alpha_j^{f_j(t_i, t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_i w_{i+1}w_{i+2})}}{Z(t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_i w_{i+1}w_{i+2})} \quad (4)$$

Equation (4) is the maximum entropy model for tagging. Each conditional probability of a prediction t_i given some context $t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_i w_{i+1}w_{i+2}$ is the product of the features that are active for that (prediction, context) pair.

Training Data The training set is created by applying the questions to each word in the training set. For example, when scanning the word *flies* in the sentence "Time flies like an arrow" the training example would be:

Prediction	Question vector
verb	w_i = flies, w_{i-1} = Time, w_{i-2} = *bd*,
	w_{i+1} = like, w_{i+2} = an,
	t_{i-1} = *noun*, $t_{i-1}t_{i-2}$ = *noun*, *bd*

Here *bd* is a special symbol for boundary. The tags have been simplified for this example; the actual tags in the Penn treebank are more fine-grained than *noun* and *verb*.

Hundreds of thousands of training samples are used to create candidate features. Any possible (prediction, question) pair that occurs in training data is a candidate feature. The feature selection strategy is a way to eliminate unreliable or noisy features from the candidate set. For the part-of-speech model described here, a simple frequency threshold is used to implement feature selection.

Given a selected feature set, the GIS algorithm is then used to find the optimal value for the corresponding α_j parameters. For this application, roughly 50 iterations of GIS sufficed to achieve convergence.

Search for Best Sequence The probability model described thus far will produce a distribution over tags, given a linguistic context including and surrounding the current word. In practice we need to tag entire sentences, which means that the model must produce a *sequence* of tags. Tagging is typically performed left-to-right, so that each decision has the left context of previously predicted tags. The probability of the best tag sequence for an n-word sentence is factored as:

$$p(t_1\ldots t_n|w_1\ldots w_n) = \prod_{i=1\ldots n} p(t_i|t_{i-1}t_{i-2}w_{i-2}w_{i-1}w_i w_{i+1}w_{i+2}).$$

The desired tag sequence is the one with the highest conditional sequence probability:

$$t_1^* \ldots t_n^* = arg_{t_1 \ldots t_n} maxp(t_1 \ldots t_n | w_1 \ldots w_n).$$

A dynamic programming procedure known as the Viterbi algorithm is typically used to find the highest probability sequence.

Other NLP Applications

Other NLP applications have used maximum entropy models to predict a wide variety of linguistic structure. The statistical parser in (Ratnaparkhi, 1999) uses separate maximum entropy models for part-of-speech, chunk, and parse structure prediction. The system in (Borthwick, 1999) uses maximum entropy models for named entity detection, while the system in (Ittycheriah, Franz, Zhu, & Ratnaparkhi, 2001) uses them as sub-components for both answer type prediction and named entity detection. Typically, such applications do not need to change the core framework, but instead need to modify the meaning of the predictions, questions, and feature selection to suit the intended task of the application.

Future Directions

Conditional random fields (Lafferty, McCallum, & Pereira, 2001), or CRFs, are an alternative to maximum entropy models that address the *label bias* issue. Label bias affects sequence models that predict one element at a time, in which features at a given state (or word, in the case of POS tagging) compete with each other, but do not compete with features at any other state in the sequence. In contrast, a CRF model has a single model for the probability of the entire sequence, and therefore allows global competition of features across the entire sequence. Parameter estimation for CRFs is related to the GIS algorithm used for maximum entropy models. See Sha & Pereira (2003) for an example of CRFs applied to noun phrase chunking.

Recommended Reading

Berger, A. L., Della Pietra, S. A., & Della Pietra, V. J. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, *22*(1), 39–71.

Borthwick, A. (1999). *A maximum entropy approach to named entity recognition*. Unpublished doctoral dissertation, New York University.

Chen, S., & Rosenfeld, R. (1999). *A Gaussian prior for smoothing maximum entropy models* (Tech. Rep. No. CMUCS-99-108). Carnegie Mellon University.

Church, K. W., & Mercer, R. L. (1993). Introduction to the special issue on computational linguistics using large corpora. *Computational Linguistics*, *19*(1), 1–24.

Curran, J., & Clark, S. (2003). Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of the 11th annual meeting of the european chapter of the association for computational linguistics (EACL'03)* (pp. 91–98). Budapest, Hungary.

Darroch, J., & Ratcliff, D. (1972). Generalized iterative scaling for log-linear models. *The Annals of Statistics*, *43*(5), 1470–1480.

Goodman, J. (2002). Sequential conditional generalized iterative scaling. In *Proceedings of the Association for Computational Linguistics (ACL)* (pp. 9–16). Philadelphia, Pennsylvania.

Ittycheriah, A., Franz, M., Zhu, W., & Ratnaparkhi, A. (2001). Question answering using maximum-entropy components. In *Proceedings of the North American association for computational linguistics. (NAACL)*, Pittsburgh, Pennsylvania.

Jaynes, E. T. (1957, May). Information theory and statistical mechanics. *Physical Review*, *106*(4), 620–630.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings 18th international conference on machine learning* (pp. 282–289). San Francisco: Morgan Kaufmann.

Lau, R., Rosenfeld, R., & Roukos, S. (1993). Adaptive language modeling using the maximum entropy principle. In *Proceedings of the ARPA Human Language Technology Workshop* (pp. 108–113). San Francisco: Morgan Kaufmann.

Malouf, R. (2002). A comparison of algorithms for maximum entropy parameter estimation. In *Sixth conference on natural language learning (CoNLL)* (pp. 49–55). Taipei, Taiwan.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1994). Building a large annotated corpus of english: "The Penn Treebank". *Computational Linguistics*, *19*(2), 313-330.

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In E. Brill & K. Church (Eds.), *Proceedings of the conference on empirical methods in natural language processing* (pp. 133–142). Somerset, NJ: Association for Computational Linguistics.

Ratnaparkhi, A. (1999). Learning to parse natural language with maximum entropy models. *Machine Learning*, *34*(1–3), 151–175.

Sha, F., & Pereira, F. (2003). Shallow parsing with conditional random fields. In *Proceedings of the human language technology conference (HLT-NAACL)* (pp. 213–220). Edmonton, Canada.

McDiarmid's Inequality

Synonyms

Bounded differences inequality

Definition

McDiarmid's inequality shows how the values of a bounded function of independent random variables

concentrate about its mean. Specifically, suppose $f : \mathcal{X}^n \to R$ satisfies the bounded differences property. That is, for all $i = 1, \ldots, n$ there is a $c_i \geq 0$ such that for all $x_1, \ldots, x_n, x' \in \mathcal{X}$

$$|f(x_1, \ldots, x_n) - f(x_1, \ldots, x_{i-1}, x', x_{i+1}, \ldots, x_n)| \leq c_i.$$

If $\mathbf{X} = (X_1, \ldots, X_n) \in \mathcal{X}^n$ is a random variable drawn according to P^n and $\mu = E_{P^n}[f(\mathbf{X})]$ then for all $\epsilon > 0$

$$P^n\left(f(\mathbf{X}) - \mu \geq \epsilon\right) \leq \exp\left(\frac{2\epsilon^2}{\sum_{i=1}^n c_i^2}\right).$$

McDiarmid's is a generalization of Hoeffding's inequality, which can be obtained by assuming $\mathcal{X} = [a, b]$ and choosing $f(\mathbf{X}) = \sum_{i=1}^n X_i$. When applied to empirical risks this inequality forms the basis of many ►generalization bounds.

MCMC

►Markov Chain Monte Carlo

MDL

►Minimum Description Length Principle

Mean Absolute Deviation

►Mean Absolute Error

Mean Absolute Error

Synonyms

Absolute error loss; Mean absolute deviation; Mean error

Definition

Mean Absolute Error is a ►model evaluation metric used with ►regression models. The mean absolute error of a model with respect to a ►test set is the mean of the absolute values of the individual prediction errors on over all ►instances in the ►test set. Each prediction error is the difference between the true value and the predicted value for the instance.

$$mae = \frac{\sum_{i=1}^n abs\left(y_i - \lambda(x_i)\right)}{n}$$

where y_i is the true target value for test instance x_i, $\lambda(x_i)$ is the predicted target value for test instance x_i, and n is the number of test instances.

Cross References

►Mean Squared Error

Mean Error

►Mean Absolute Error

Mean Shift

Xin Jin, Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, IL, USA

Mean shift (Comaniciu & Meer, 2002) is a nonparametric algorithm for ►partitional clustering which does not require specifying the number of clusters, and can form any shape of clusters.

Given n data points $x_i, i = 1, \ldots, n$, in the d-dimensional space R^d, the multivariate kernel density estimator obtained with kernel $K(x)$ and window radius h is given by

$$f(x) = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right). \quad (1)$$

Given the gradient of the density estimator, the mean shift is defined as the difference between the weighted (using the kernel as weights) mean and x, the center of the kernel,

$$m_h(x) = \frac{\sum_{i=1}^n x_i g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)} - x. \quad (2)$$

The mean shift vector is proportional to the normalized density gradient estimate, and thus points to the

direction of the maximum increase in the density. By successively computing the mean shift vector and translating the kernel (window) by the vector, the mean shift procedure can guarantee converging at a nearby point where the gradient of density function is zero.

Recommended Reading

Comaniciu, D., & Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions of the Pattern Analysis and Machine Intelligence, 24*(5), 603–619.

Mean Squared Error

Synonyms

Quadratic loss; Squared error loss

Definition

Mean Squared Error is a ►model evaluation metric often used with ►regression models. The mean squared error of a model with respect to a ►test set is the mean of the squared prediction errors over all ►instances in the ►test set. The prediction error is the difference between the true value and the predicted value for an instance.

$$mse = \frac{\sum_{i=1}^{n} (y_i - \lambda(x_i))^2}{n}$$

where y_i is the true target value for test instance x_i, $\lambda(x_i)$ is the predicted target value for test instance x_i, and n is the number of test instances.

Cross References

►Mean Absolute Error

Measurement Scales

Ying Yang
Australian Taxation Office, Box Hill, Australia

Definition

Turning to the authority of introductory statistical textbooks (Bluman, 1992; Samuels & Witmer, 1999), there are two parallel ways to classify data into different types. Data can be classified into either ►categorical or ►numeric. Data can also be classified into different levels of ►measurement scales.

There are two parallel ways to classify data into different types. Data can be classified into either categorical or numeric. Data can also be classified into different *levels of* measurement scales.

Categorical versus Numeric

Variables can be classified as either categorical or numeric. **Categorical variables**, also often referred to as **qualitative variables**, are variables that can be placed into distinct categories according to some characteristics. Categorical variables sometimes can be arrayed in a meaningful rank order. But no arithmetic operations can be applied to them. Examples of categorical variables are

- Gender of a fish: male and female
- Student evaluation: fail, pass, good, and excellent

Numeric variables, also often referred to as *quantitative variables*, are numerical in nature. They can be ranked in order. They can also have meaningful arithmetic operations. Numeric variables can be further classified into two groups, discrete or continuous.

A *discrete variable* assumes values that can be counted. The variable cannot assume all values on the number line within its value range. An example of a discrete variable is *the number of children in a family*.

A *continuous variable* can assume all values on the number line within the value range. The values are obtained by measuring. An example of a continuous variable is *Fahrenheit temperature*.

Levels of Measurement Scales

In addition to being classified as either categorical or numeric, variables can also be classified by how they are categorized, counted, or measured. This type of classification uses measurement scales, and four common types of scales are used: nominal, ordinal, interval, and ratio.

The *nominal* level of measurement scales classifies data into mutually exclusive (nonoverlapping), exhaustive categories in which no order or ranking can be imposed on the data. An example of a nominal variable is *gender of a fish*: male and female.

The *ordinal* level of measurement scales classifies data into categories that can be ranked. However, the differences between the ranks cannot be calculated by arithmetic. An example of an ordinal variable is *student*

evaluation, with values fail, pass, good, and excellent. It is meaningful to say that the student evaluation of pass ranks is higher than that of fail. It is not meaningful in the same way to say that the gender female ranks higher than the gender male.

The **interval** level of measurement scales ranks the data, and the differences between units of measure can be calculated by arithmetic. However, *zero* in the interval level of measurement means neither "nil" nor "nothing" as *zero* in arithmetic means. An example of an interval variable is *Fahrenheit temperature*. It is meaningful to say that the temperature A is two points higher than the temperature B. It is not meaningful in the same way to say that the student evaluation of pass is two points higher than that of fail. Besides, 0°F does not mean the absence of heat.

The **ratio** level of measurement scales possesses all the characteristics of interval measurement, and there exists a *zero* that, the same as arithmetic *zero*, means "nil" or "nothing." In consequence, true ratios exist between different units of measure. An example of a ratio variable is *number of children in a family*. It is meaningful to say that the number of children in the family A is twice that of the family B. It is not meaningful in the same way to say that the Fahrenheit temperature A is twice that of B.

The nominal level is the lowest level of measurement scales. It is the least powerful in terms of including data information. The ordinal level is higher. The interval level is even higher. The ratio level is the highest level. Any data conversion from a higher level of measurement scales to a lower level of measurement scales, such as ►discretization, will lose information. Table 1 gives a summary of the characteristics of different levels of measurement scales.

Measurement Scales. Table 1 Characteristics of different levels of measurement scales

Level	Ranking?	Arithmetic Operation?	Arithmetic Zero?
Nominal	No	No	No
Ordinal	Yes	No	No
Interval	Yes	Yes	No
Ratio	Yes	Yes	Yes

Summary

In summary, the following taxonomy applies to variable types:

- Categorical (qualitative) variables:
  - Nominal
  - Ordinal
- Numeric (quantitative) variables:
  - Interval, either discrete or continuous
  - Ratio, either discrete or continuous

Recommended Reading

Bluman, A. G. (1992). *Elementary statistics: A step by step approach*. Wm. C. Brown Publishers Dubuque, Iowa, USA.

Samuels, M. L. & Witmer, J. A. (1999). *Statistics for the life sciences* (2nd ed.). Upper Saddle River, NJ: Prentice-Hall Publishers, USA.

Medicine: Applications of Machine Learning

Katharina Morik
Technische Universität Dortmund, Dortmund, Germany

Motivation

Health care has been an important issue in computer science since the 1960s. In addition to databases storing patient records, library resources (e.g., PubMed, a service of the U.S. National Library of Medicine that includes over 16 million citations from journals for biomedical articles back to the 1950s), administrative and financial systems, more sophisticated support of health care has been the aim of artificial intelligence (AI) from the very beginning on. Starting with expert systems which abstract laboratory findings and other vital parameters of a patient before they heuristically classify the patient into one of the modeled diagnoses (Shortliffe, 1976), knowledge acquisition was discovered to be the bottleneck of systems for the automatic medical diagnosis. Machine learning came into play as a means of knowledge acquisition. Learning rules for (medical) expert systems focused on the heuristic classification step within expert systems. Given conveniently abstracted measurements of the patient's state, the classification was learned in terms of rules or ►decision

trees. Since the early days, the use of machine learning for health care progressed in two ways:

- The abstraction of measurements of a patient's vital parameters is a learning task in its own right. Diverse kinds of data are to be handled: laboratory data, online measurements at the bedside, x-rays or other imaging data, genetic data,... Machine learning is confronted with a *diversity of representations* for the examples.
- Diagnosis is just one task in which physicians are to be supported. There are many more tasks which machine learning can ease. In intensive care, the addressee of the learning results can be a machine, e.g., the respirator. Financing health care and planning the medical resources (e.g., for a predicted epidemia) are yet another important issue. Machine learning is placed in a *diversity of medical tasks*.

The urgent need for sophisticated support of health care follows from reports which estimate up to 100,000 deaths in the USA each year due to medical error (Kohn, Corrigan, & Donaldson, 2000).

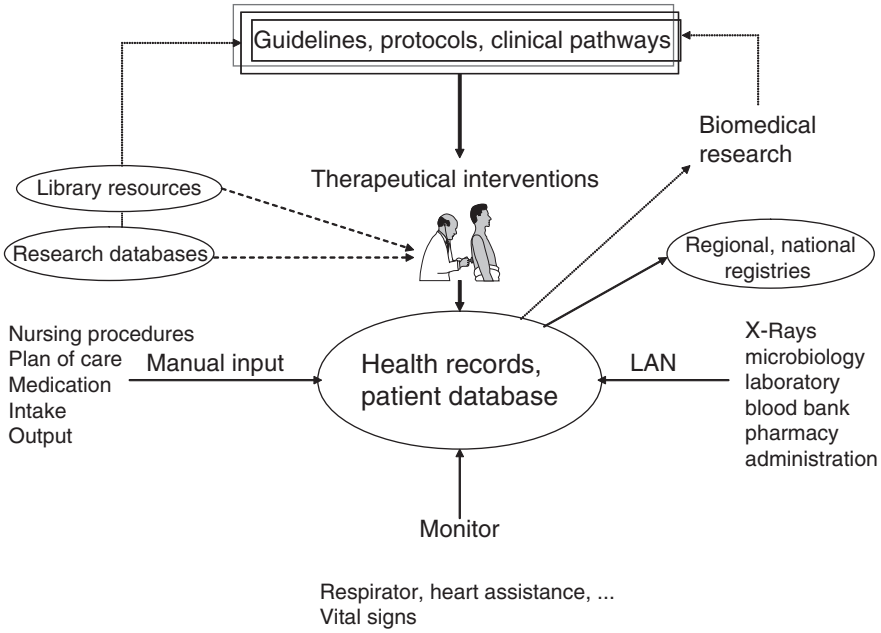
Structure of the Problem

The overall picture of the medical procedures shows the kinds of data and how they are entered into the database of health records (a synonym is "patient database.") A monitoring system is given in intensive care units, which acquires ▶time series from minute measurements. The observations at the bedside are entered manually into the system. The information from the hospital is entered via a local area network. The physician accesses information from libraries and research databases (dashed lines). Libraries, research databases, and biomedical research also influence the development of guidelines, protocols, and clinical pathways (dotted lines). Guidelines are rather abstract. Protocols of certain actions are integrated to become a clinical pathway which is a plan of both diagnostic and therapeutical actions for a typical patient with a specific diagnosis. The bold arrow shows the intended high-quality therapy. Guidelines and protocols promote evidence-based practices, reduce inter-clinician practice variations and support decision-making in patient care while constraining the costs of care. Computerized protocols can be generated based on guidelines. They have been proved useful in improving the quality and consistency of healthcare but the protocol development process is time-consuming (Ten Teije, Lucas, & Miksch, 2006). This is where machine learning offers support. Usually, ontologies (e.g., in description logic) or other knowledge-based techniques (in medicine-specific formats like the Arden Syntax, GuideLine Interchange Format (GLIF), PROforma, Asbru, and EON) are used to support the development of protocols (de Clercq, Blomb, Korstenb, & Hasman, 2004). By contrast, machine learning induces the current practices and their outcome from the health records (Smith, Doctor, Meyer, Kalet & Philips, 2009). To reflect such use of Machine Learning, the bold arrows of the picture would need to be turned the other way around, protocols are learned from the data or evaluated based on the data. All (reversed) arrows mark possible applications of machine learning.

Diversity of Representations

The overall health record of a patient includes several types of data, not all of them are digital.

- Laboratory data consist of attributes almost always with numerical values, sometimes with discrete ordinal values, sometimes just binary values like "positive," "negative."
- Plain text states anamneses, diagnosis, and observations. From the text, key words can be transformed into attributes for machine learning.
- Online measurements at the bedside are time series. They are analyzed in order to find level changes or trends (Gather, Schettlinger, & Fried, 2006) and alarm the physician (Sieben & Gather, 2007). In order to exploit the time series for further learning tasks, they often are abstracted (e.g., Bellazzi, Larizza, Magni, & Bellazi (2002)). Recently, online measurements from body sensors have raised attention in the context of monitoring patients at home (Amft & Tröster, 2008).
- Sequences can also be considered time series, but the measurements are not equidistant and not restricted to numerical values. Examples are data gathered at doctors' visits and long-term patient observations.
- X-rays or other imaging data (e.g., ultrasound imaging or more novel molecular imaging techniques like positron emission tomography, magnetic resonance imaging, or computer tomography) cannot be



analyzed directly by machine learning algorithms. They require the extraction of features. It has been shown that the adequate extraction of features is more important than the selection of the best suited learning algorithm (Mavroforakis, Georgiou, Dimitropoulos, Cavouras, & Theodoridis, 2006). The number of extracted features can become quite large. For instance, from 1,619 images of skin lesion, each 752 × 582 pixels, 107 features were extracted in order to detect melanoma using diverse learning algorithms (Dreiseitl et al., 2001). Hence, feature selection is also an important task in medical applications (Lucaces, Taboada, Albaiceta, Domingues, Enriques & Bahamonde, 2009; Withayachumnankul et al., 2006). Often, different techniques are applied to gather data for the detection of the same disease. For instance, glaucoma detection uses standard automated perimetry or scanning laser or Heidelberg Retina Tomograph or stratus optical coherence tomography. It is not yet clear how important the choice among measurement types (devices) is with respect to feature extraction and machine learning.

- Tissue and blood: In vitro "data" also belong to health records. Immediately after biopsy or surgery, the tissue is transferred to the pathology department. After the pathologist has taken the sample needed for proper diagnosis, a representative tissue sample will be snap frozen and stored in liquid nitrogen or at −80∘C. Also blood cells are stored in a blood bank. From the specimen, the RNA is extracted and the so-called microarrays of gene expressions are developed and then scaled. The huge prognostic value of gene expression in patients with breast cancer has been shown by van't Veer et al. (2002). Genome research aims at revealing the impact of gene regulation and protein expression-regulation (taking into account the regulation of protein synthesis, protein ubiquitination, and post-translational modification) on, e.g., cancer diagnosis and response to therapies. Machine learning, particularly clustering, frequent itemset mining, and classification have been applied successfully (see ►learning from gene expression microarray data).

In addition to patient records, there are knowledge bases describing particular diseases or computerized protocols for particular therapies.

Medical Tasks

Diagnosis and Medication

Diagnosis is primarily a classification task. Given the description of the patient's state and a set of diseases, the learning algorithm outputs the classification

into one of the classes. If physicians want to inspect the learned classifier, logic-based algorithms are preferred. Decision trees and the conceptual clustering algorithm AQ were used to diagnose breast cancer from nine abstracted descriptions like `tumor size`: 0 – 4, 5 – 9, ⋯, 50 – 54, 55 – 59 (Cestnik, Kononenko, & Bratko, 1987; Michalski, Mozetic, Hong, & Lavrac, 1986).

►Bayesian methods were used to classify, e.g., diseases of the lymph node. Based on the examination of the extracted tissue, a pathologist enters the description. The Bayesian network (BN) outputs not only just one diagnosis, but the conditional probabilities for the diseases (Heckerman, 1990). In particular, diagnosis for rather vague symptoms such as abdominal pain or lower back pain is well supported by BNs (McNaught, Clifford, Vaughn, Foggs, & Foy, 2001). BNs are capable of incorporating given expert knowledge as priors. In order to combine textbook knowledge with empirical data, electronic literature was transformed into priors for BN structures. Then, from health records, the BN was learned as a model of ovarian tumors (Antal, Fannes, Timmerman, Moreau, & De Moor, 2004).

►Inductive logic programming (ILP) also allows to take into account background knowledge. This was used for an enhanced learning of medical diagnostic rules (Lavrac, Dzeroski, Prinat, & Krizman, 1993). The identification of glaucomatous eyes was effectively learned by ILP (Mizoguchi, Ohwada, Daidoji, & Shirato, 1997). One advantage of ILP is that the learned logic clauses can easily be integrated into a knowledge-based system and, hence, become operational for clinical practice.

Since some tests which deliver information about the patient's state can be costly – both, financially and in terms of a risk for the patient – ►cost-sensitive learning may be applied.

Since the error of classifying a patient as ill where he or she is not (false positives) is less harmful than classifying a patient as healthy where he or she is not (false negatives), the evaluation of the learning result most often is used in a biased way. The evaluation can be summarized in Table 1.

Precision is the proportion $\frac{A}{A+B}$, and *recall* is the proportion $\frac{A}{A+C}$. *Sensitivity* is synonymous to recall. In medical applications, sensitivity is balanced with respect to *specificity* being the proportion $\frac{B}{B+D}$ (synonym *false positives rate*). The analysis of the Receiver Operator Characteristic (ROC) allows to evaluate learning according to sensitivity and specificity (see ►ROC analysis).

Medicine: Applications of Machine Learning. Table 1
Evaluation measures

	True +	False –
Predicted +	A	B
Predicted –	C	D

If not the understandability but only sensitivity and specificity are important, numerical learning algorithms are used to classify the patient's data. In particular, if the patient's state is described by numerical features, no discretization is necessary for numerical learners as is needed for the logic-based ones. Multilayer perceptrons (see ►Neural Networks), ►support vector machines (SVM), ►mixtures of Gaussians, and mixture of generalized Gaussian classifiers were trained on the numerical data of 189 normal eyes and 156 glaucomatous eyes (Goldbaum et al., 2002). The numerical description of the visual field is given by standard automated threshold perimetry. The medical standard procedure to interpret the visual field is to derive global indices. The authors compared performance of the classifiers with these global indices, using the area under the ROC curve. Two human experts were judged against the machine classifiers and the global indices by plotting their sensitivity–specificity pairs. The mixture of Gaussian had the greatest area under the ROC curve of the machine classifiers, and human experts were not better at classifying visual fields than the machine classifiers or the global indices.

Other approaches to glaucoma detection use different features describing the patient's state (Zangwill et al., 2004) or other numerical learners, e.g., ►logistic regression (Huang, Chen, & Hung, 2006). For testing the learning from numerical attributes, the UCI Machine Learning Repository offers the arrhythmia database. The aim is to distinguish between the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups. About 279 attributes are given, 206 of them being numerical ones.

As has been shown in an application to intensive care, medication can be transformed into a set of classification tasks (Morik, Imhoff, Brockhausen, Joachims,

& Gather, 2000). Given measurements of eight vital signs, a decision is made for each of six drugs, whether to increase or to decrease it. This gives a set of classification tasks, which the ►SVM learned. Depending on the drug, the accuracy ranged from 81.3% with 2.5 standard error to 86.9% with 7 standard error. Additionally, on 41 cases, the SVM decision was compared with an expert's decisions when confronted with the same data. In 32 cases the expert chose the same direction of change as did the learned decision function. In 34 cases the learned decision was equal to the actual therapy. Another set of classification tasks were to decide every minute whether to increase, decrease, or leave the doses as it is. Again, each of these classifiers was learned by the SVM. From 1,319 examples decision functions were learned and tested on 473 examples. For training, an unbalanced cost function was used. The SVM cost factor for error was chosen according to $\frac{C_+}{C_-} = \frac{number\ of\ negative\ examples}{number\ of\ positive\ examples}$. The results again differed depending on the drug. For adrenaline, 79% of the test cases were equally handled by the physician and the decision function. For adrenaline as well as for dobutamine, only in 1.5% of the test cases the learned rule recommended the opposite direction of change. Again, a blind test with an expert showed that the learned recommendations' deviation from actual therapy was comparable to that of the human expert. Combining the two sets of classifications, for each minute and each patient, the support vector machine's decision function outputs a recommendation of treatment (Morik, Imhoff, Brockhausen, Joachims, & Gather, 2000).

Prognosis and Quality of Care Assessment

Prognosis or outcome prediction is important for the evaluation of the quality of care provided. The standard statistical models use only a small set of covariates and a score variable, which indicates the severity of the illness. Machine learning may also rely on the aggregated score features, but is in addition capable of handling the features underlying the scores. Given health records of patients including the therapy, machine learning is to predict the outcome of care, e.g., classifies into mortal or surviving cases. The prediction of breast cancer survivability has been tested on a very large database comparing three learning methods (Delen, Walker, & Kadam, 2004). The results indicated that decision trees (here: C5) result in the best predictor with 93.6% accuracy on the holdout sample (this prediction accuracy is better than any reported in the literature), artificial neural networks came out to be the second with 91.2% accuracy, and the ►logistic regression models came out to be the worst of the three with 89.2% accuracy.

Prediction of survival is a hard task for patients with serious stroke, because there is a long-term risk after the stay at the hospital. The scoring schemes (e.g., the Glasgow coma scale and the Ranking score) are not sufficient for predicting the outcome. In a data situation where 29 attributes (or features) were given for only 327 patient records, BNs were learned and compared with a handmade causal network. The results were encouraging – as soon as more electronic health records become available, the BNs will become closer to medical knowledge. Moreover, the discovery of relations on the basis of empirical data may enhance medical knowledge (Wu, Lucas, Kerr, & Dijkhuisen, 2001).

Carcinogenesis prediction was performed using ILP methods. As has become usual with cancer diagnosis and prognosis, there is a close link with microbiology (Srinivasan, Muggleton, King, & Sternberg, 1994)(see Learning from gene expression microarray data).

Prognosis need not be restricted to mortality rates. In general, it is a means of quality assessment of clinical treatments. For instance, hemodialysis services have been assessed through temporal data mining by Bellazzi et al. (2002).

Finding subgroups of patients with devious reactions to a therapy might lead to a better understanding of a certain medical process (Atzmueller, Baumeister, Hensing, Richter, & Puppe, 2005). While the before mentioned study aims at an enhanced expert – system interaction, a Dutch study aims at a more precise modeling of prognoses (Abu-Hanna & Lucas, 2001). In an extensive study for eight different hospitals and 7,803 patients, two different models were combined: one for determining the subgroups and the other for building a model for each subgroup. For the prognoses of patients in an intensive care unit, subgroups have been detected using decision trees. The decision tree was trained to classify patients into the survival class and the mortality class on the basis of the nonaggregated features underlying the illness score. The leaves of the

tree become subgroups. These are then used for training a logistic regression model of mortality based on the aggregated features.

Verification and Validation

Verification is the process of testing a model against a specification. In medicine, this often means to check clinical practice against expert protocols, or to check an actual diagnosis against one derived from textbook knowledge. Since many logic-based machine learning algorithms consist of a generalization and a specialization step, they can be used for verification. Generalization delivers rules from clinical data which can then be compared with given expert rules (protocols). Specialization is triggered by facts that contradict a learning hypothesis. Hence, using an expert rule as hypothesis, the learning algorithm counts the contradicting clinical cases and specializes the rule. For an early case study on verification and rule enhancement see, e.g., (Morik, Potamias, Moustakis, & Charissis, 1994). A more recent study compares a given clinical protocol for intensive care with actual therapies at another hospital (Scholz, 2002). Decision trees and association rules have been learned in order to inspect and enhance the knowledge base of a web-based teledermatology system (Ou, West, Lazarescu, & Clay, 2007). While verification means to build the system right, validation means to build the right system. The borderline between verification and validation is fuzzy. On the one hand, medical practice is investigated with respect to the guidelines (verification), on the other hand, the guidelines are enhanced on the basis of medical practice (validation).

Moreover, learned models can be verified with respect to expert knowledge and validated with respect to clinical practice. A study on the hemodynamic monitoring of the critically ill integrated machine learning into a knowledge-based approach to evidence-based medicine. A knowledge base on drug effects was verified using patient records. Only 18% of the observations showed vital signs of patients in the opposite direction than predicted by the knowledge base. Then, the knowledge base was used to validate therapeutical interventions proposed by a learned model. Accuracy measures of a model only reflect how well the learning result fits actual behavior of the physician and not how well it fits the "gold standard." Hence, a proposed intervention should be validated with respect to its effects on the patient. If the known effects push vital signs in the direction of the desired value range, the recommendation is considered sound, otherwise it is rejected. Using past data, the learned model was found to recommend an intervention with the desired effects in 81% of the cases (Morik, Joachims, Imhoff, Brockhausen, & Rüping, 2002).

Intelligent Search in Medical Literature

Intelligent search in the overwhelming number of research publications supplies the information when it is needed. ILP has been successfully put to use for finding relevant medical documents (Dimec, Dzeroski, Todorovski, & Hristovski, 1999). Also the intelligent search in clinical free-text guidelines is an issue (Moskovitch et al., 2006). The techniques for text categorization can be applied to medical texts in the usual way. If the search engine not only labels the overall document but, in addition, phrases within it, the search could become more focused and also deliver paragraphs instead of complete texts. The biomedical challenge for *named entity recognition* requires the automatic extraction and classification of words referring to DNA, RNA, proteins, cell types, and cell lines from texts (Kim, Ohta, Tsuruoka, Tateisi, & Collier, 2004). Even more difficult is the discovery of medical knowledge from texts (Sanchez & Moreno, 2005).

Epidemiology and Outbreak Detection

Understanding the transmission of infectious diseases and forecasting epidemics is an important task, since infections are distributed globally. Statistical approaches to spatio-temporal analysis of scan data are regularly used. There, a grid partitions the map into regions where occurrences of the disease are shown as points. "Hot spot" partitions are those of high density. By contrast, clustering detects hot spot regions depending on the data, hence, the shape of regions is flexible. Taking into account the a priori density of the population, a risk-adjusted nearest neighbor hierarchical clustering discovers "hot spot" regions. Also a risk-adjusted support vector machine with Gaussian kernel has successfully been applied to the problem of detecting regions with infectious disease outbreak. The

discovery of hot spot regions can be exploited for predicting virus activity, if an indicator is known which can easily be observed. For instance, dead crows indicate activity of the West Nile virus. An overview of infectious disease informatics is given by (Zeng, Chen, Lynch, Eidson, & Gotham, 2005).

Machine learning can also contribute to the understanding of the transmission of infectious diseases. A case study on tuberculosis epidemiology uses BNs to identify the distribution of tuberculosis patient attributes. The learning results captured the known statistical relationships. A relational model learned from the database directly using structured statistical models revealed several novel associations (Getoor, Rhee, Koller, & Small, 2004).

Cross References

- ▶Class Imbalance Problem
- ▶Classification
- ▶Classifier Systems
- ▶Cost-Sensitive Learning
- ▶Decision Trees
- ▶Feature Selection
- ▶Inductive Logic Programming
- ▶ROC Analysis
- ▶Support Vector Machine
- ▶Time Series

Recommended Reading

Abu-Hanna, A., & Lucas, P. J. F. (2001). Prognostic models in medicine: AI and statistical approaches [Editorial]. *Methods of Information in Medicine*, *40*(1), 1–5.

Amft, O., & Tröster, G. (2008). Recognition of dietary events using on-body sensors. *Artifical Intelligence in Medicine, 42*(2), 121–136.

Antal, P., Fannes, G., Timmerman, D., Moreau, Y., & De Moor, B. (2004). Using literature and data to learn BNs as clinical models of ovarian tumors. *Artificial Intelligence in Medicine, 30*(3), 257–281.

Atzmueller, M., Baumeister, J., Hensing, A., Richter, E.-J., & Puppe, F. (2005). Subgroup mining for interactive knowledge refinement. In *Artificial intelligence in medicine (AIME)* (pp. 453–462). Berlin/Heidelberg: Springer.

Bellazzi, R., Larizza, C., Magni, P., & Bellazi, R. (2002). Quality assessment of dialysis services through intelligent data analysis and temporal data mining. In *Workshop at the 15th European conference on AI about intelligent data analysis in medicine and pharmacology* (pp. 3–9). Lyon, France.

Cestnik, B., Kononenko, I., & Bratko, I. (1987). ASSISTANT 86: A knowledge-elicitation tool for sophisticated users. In I. Bratko, & N. Lavrac (Eds.), *Progress in machine learning* (pp. 31–45). Wilmslow, GB: Sigma Press.

de Clercq, P. A., Blomb, J. A., Korstenb, H. H., & Hasman, A. (2004). Approaches for creating computer-interpretable guidelines that facilitate decision support. *Artificial Intelligence in Medicine, 31*(1), 1–27.

Delen, D., Walker, G., & Kadam, A. (2004). Predicting breast cancer survivability: A comparison of three data mining methods. *Artificial Intelligence in Medicine, 34*(2), 113–127.

Dimec, B., Dzeroski, S., Todorovski, L., & Hristovski, D. (1999). WWW search engine for slovenian and english medical documents. In *Proceedings of the 15th international congress for medical informatics* (pp. 547–552). Amsterdam: IOS Press.

Dreiseitl, S., Ohn-Machado, L., Kittler, H., Vinterbo, S., Billhardt, H., & Binder, M. (2001). A comparison of machine learning methods for the diagnosis of pigmented skin lesions. *Journal of Biomedical Informatics, 34*, 28–36.

Gather, U., Schettlinger, K., & Fried, R. (2006). Online signal extraction by robust linear regression. *Computational Statistics, 21*(1), 33–51.

Getoor, L., Rhee, J. T., Koller, D., & Small, P. (2004). Understanding tuberculosis epidemiology using structured statistical models. *Artificial Intelligence in Medicine, 30*(3), 233–256.

Goldbaum, M. H., Sample, P. A., Chan, K., Williams, J., Lee, T-W., Blumenthal, E., et al. (2002). Comparing machine learning classifiers for diagnosing glaucoma from standard automated perimetry. *Investigative Ophthalmology and Visual Science, 43*, 162–169.

Heckerman, D. (1990). *Probabilistic similarity networks*. Technical report STAN-CS-1316, Department of Computer Science and Medicine at Stanford.

Huang, M. L., Chen, H. Y., & Hung, P. T. (2006). Analysis of glaucoma diagnosis with automated classifiers using stratus optical coherence tomography. *Optical Quantum Electronics, 37*, 1239–1249.

Kim, J. D., Ohta, T., Tsuruoka, Y., Tateisi, Y., & Collier, N. (2004). Introduction to the bio-entity recognition task at JNLPBA. In N. Collier, P. Ruch, & A. Nazarenko, (Eds.), *Proceedings of the international joint workshop on natural language processing in biomedicine and its applictions* (pp. 70–76). Morristown, NJ: ACL.

Kohn, L. T., Corrigan, J. M., & Donaldson, M. (Eds.) (2000). *To err is human – building a safer health system*. Washington, DC: National Academic Press.

Lavrac, N., Dzeroski, S., Prinat, V., & Krizman, V. (1993). The utility of background knowledge in learning medical diagnostic rules. *Applied Artificial Intelligence, 7*, 273–293.

Lucaces, O., Taboada, F., Albaiceta, G., Domingues, L. A., Enriques, P., & Bahamonde, A. (2009). Predicting the probability of survival in intensive care unit patients from a small number of variables and training examples. *Artificial Intelligence in Medicine, 45*(1), 63–76.

Mavroforakis, M., Georgiou, H., Dimitropoulos, N., Cavouras, D., & Theodoridis, S. (2006). Mammographic masses characterization based on localized texture and dataset fractal analysis using linear, neural and support vector machine classifiers. *Artificial Intelligence in Medicine, 37*(2), 145–162.

McNaught, K., Clifford, S., Vaughn, M., Foggs, A., & Foy, M. (2001). A Bayesian belief network for lower back pain diagnosis. In P. Lucas, L. C. van der Gaag, & A. Abu-Hanna (Eds.), *Bayesian models in medicine – Workshop at AIME*. Caseais, Portugal.

Michalski, R., Mozetic, I., Hong, J., & Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application on three medical domains. In *Proceedings of the 5th national conference on artificial intelligence* (pp. 1041–1045). San Mateo,CA: Morgan Kaufmann.

Mizoguchi, F., Ohwada, H., Daidoji, M., & Shirato, S. (1997). Using inductive logic programming to learn classification rules that identify glaucomatous eyes. In N. Lavrač, E. Keravnou, & B. Zupan, (Eds.), *Intelligent data analyis in medicine and pharmacology* (pp. 227–242). Norwell, MA: Kluwer.

Morik, K., Imhoff, M., Brockhausen, P., Joachims, T., & Gather, U. (2000). Knowledge discovery and knowledge validation in intensive care. *Artificial Intelligence in Medicine, 19*(3), 225–249.

Morik, K., Joachims, T., Imhoff, M., Brockhausen, P., & Rüping, S. (2002). Integrating kernel methods into a knowledge-based approach to evidence-based medicine. In M. Schmitt, H. N. Teodorescu, A. Jain, A. Jain, S. Jain, & L. C. Jain, (Eds.), *Computational intelligence processing in medical diagnosis*, (Vol. 96) Studies in fuzziness and soft computing, (pp. 71–99). New York: Physica-Verlag.

Morik, K., Potamias, G., Moustakis, V. S., & Charissis, G. (1994). Knowledgeable learning using MOBAL: A medical case study. *Applied Artificial Intelligence, 8*(4), 579–592.

Moskovitch, R., Cohen-Kashia, S., Drora, U., Levya, I., Maimona, A., & Shahar, Y. (2006). Multiple hierarchical classification of free-text clinical guidelines. *Artificial Intelligence in Medicine, 37*(3), 177–190.

Ou, M., West, G., Lazarescu, M., & Clay, C. (2007). Dynamic knowledge validation and verification for CBR teledermatology system. *Artificial Intelligence in Medicine, 39*(1), 79–96.

Sanchez, D., & Moreno, A. (2005). Web mining techniques for automatic discovery of medical knowledge. In *Proceedings of the 10th conference on artificial intelligence in medicine*. Aberdeen, Scotland.

Scholz, M. (2002). Using real world data for modeling a protocol for ICU monitoring. In P. Lucas, L. Asker, & S. Miksch, (Eds.), *Working notes of the IDAMAP 2002 workshop*, (pp. 85–90). Lyon, France.

Shipp, M. A., Ross, K. N., Tamayo, P., Weng, A. P., Kutok, J. L., Aguiar, R. C., et al. (2002). Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Nature Medicine, 8*(1), 68–74.

Shortliffe, E. H. (1976). *Computer based medical consultations: MYCIN*. New York, Amsterdam: Elsevier.

Sieben, W., & Gather, U. (2007). Classifying alarms in intensive care–analogy to hypothesis testing. In *11th conference on artifical intelligence in medicine (AIME)* (pp. 130–138). Berlin: Springer.

Smith, W. P., Doctor, J., Meyer, J., Kalet, I. J., & Philips, M. H. (2009). A decision aid for intensity-modulated radiation-therapy plan selection in prostate cancer based on a prognostic Bayesian network and a Markov model. *Artificial Intelligence in Medicine, 46*(2), 119–130.

Srinivasan, A., Muggleton, S. H., King, R. D., & Sternberg, M. J. E. (1994). Carcinogenesis prediction using inductive logic programming. In B. Zupan, E. Keravnou, & N. Lavrac (Eds.), *Intelligent data analysis in medicine and pharmacology* (pp. 243–260). Norwell, MA: Kluwer.

Ten Teije, A., Lucas, P., & Miksch, S. (Eds.), (2006). *Workshop on AI techniques in healthcare: Evidence-based guidelines and protocols, held in conjunction with ECAI-2006*. Italy.

van't Veer, L. J., Dai, H. Y., van de Vijver, M. J., He, Y. D. D., Hart, A. A., Mao, M., et al. (2002). Gene expression profiling predicts clinical outcome of breast cancer. *Nature, 415*, 530–536.

Withayachumnankul, W., Ferguson, B., Rainsford, T., Findlay, D., Mickan, S. P., & Abbott, D. (2006). T-ray relevant frequencies for osteosarcoma classification. In D. Abbott, Y. S. Kivshar, H. H. Rubinstein-Dunlop, & S.-H. Fan, (Eds.), *Proceedings of SPIE*. Brisbane, Australia.

Wu, X., Lucas, P., Kerr, S., & Dijkhuisen, R. (2001). Learning bayesian-network topologies in realistic medical domains. In *Intelligent data analysis in medicine and pharmacology*. Medical Data Analysis, (pp. 302–307). Berlin/Heidelberg: Springer.

Zangwill, L. M., Chan, K., Bowd, C., Hao, J., Lee, T. W., Weinreb, R. N., et al. (2004). Heidelberg retina tomograph measurements of the optic disc and parapillary retina for detecting glaucoma analyzed by machine learning classifiers. *Investigative Ophthalmology and Visual Science, 45*(9), 3144–3151.

Zeng, D., Chen, H., Lynch, C., Eidson, M., & Gotham, I. (2005). Infectious disease informatics and outbreak detection. In H. Chen, S. Fuller, C. Friedman, & W. Hersh, (Eds.), *Medical informatics: knowledge management and data mining in biomedicine* (pp. 359–395). New York: Springer.

Memory Organization Packets

►Dynamic Memory Model

Memory-Based

►Instance-Based Learning

Memory-Based Learning

►Case-Based Reasoning

Merge-Purge

►Entity Resolution

Message

In ►Minimum Message Length inference, a binary sequence conveying information is called a message.

Meta-Combiner

A *meta-combiner* is a form of ►ensemble learning technique used with ►missing attribute values. Its common topology involves base learners and classifiers at the first level, and meta-learner and meta-classifier at the second level. The meta-classifier combines the decisions of all the base classifiers.

Metaheuristic

Marco Dorigo, Mauro Birattari,
Thomas Stützle

A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework that can be applied to different optimization problems with relatively few modifications. Examples of metaheuristics include simulated annealing, tabu search, iterated local search, evolutionary algorithms, and ant colony optimization.

Metalearning

Pavel Brazdil[1], Ricardo Vilalta[2],
Christophe Giraud-Carrier[3], Carlos Soares[1]
[1]University of Porto, Porto, Portugal
[2]University of Houston,
Houston TX, USA
[3]Brigham Young University, UT, USA

Synonyms

Adaptive learning; Dynamic selection of bias; Learning to learn; Ranking learning methods; self-adaptive systems

Definition

Metalearning allows machine learning systems to benefit from their repetitive application. If a learning system fails to perform efficiently, one would expect the learning mechanism itself to adapt in case the same task is presented again. Metalearning differs from base-learning in the scope of the level of adaptation; whereas learning at the base-level is focused on accumulating experience on a specific task (e.g., credit rating, medical diagnosis, mine-rock discrimination, fraud detection, etc.), learning at the metalevel is concerned with accumulating experience on the performance of multiple applications of a learning system.

Briefly stated, the field of metalearning is focused on the relation between tasks or domains, and learning algorithms. Rather than starting afresh on each new task, metalearning facilitates evaluation and comparison of learning algorithms on many different previous tasks, establishes benefits and disadvantages, and then recommends the learning algorithm, or combination of algorithms that maximizes some utility function on the new task. This problem can be seen as an algorithm selection task (Rice, 1976).

The utility or usefulness of a given learning algorithm is often determined through a mapping between characterization of the task and the algorithm's estimated performance (Brazdil & Henery, 1994). In general, metalearning can recommend more than one algorithm. Typically, the number of recommended algorithms is significantly smaller than the number of all possible (available) algorithms (Brazdil, Giraud-Carrier, Soares, & Vilalta, 2009).

Motivation and Background

The application of machine learning systems to ►classification and ►regression tasks has become a standard, not only in research but also in commerce and industry (e.g., finance, medicine, and engineering). However, most successful applications are custom-designed, the result of skillful use of human expertise. This is due, in part, to the large, ever increasing number of available machine learning systems, their relative complexity, and the lack of systematic methods for discriminating among them. The problem is further compounded by the fact that, in ►Knowledge Discovery from Databases, each operational phase (e.g., preprocessing, model generation) may involve a choice among various possible alternatives (e.g., progressive vs. random sampling, neural network vs. decision tree learning), as observed by Bernstein, Provost, and Hill (2005).

Current data mining systems are only as powerful as their users. These tools provide multiple algorithms within a single system, but the selection and combination of these algorithms must be performed before the system is invoked, generally by an expert user. For some researchers, the choice of learning and data transformation algorithms should be fully automated if machine learning systems are to be of any use to nonspecialists. Others claim that full automation of the data mining process is not within the reach of current technology. An intermediate solution is the design of assistant systems aimed at helping to select the right learning algorithm(s). Whatever the proposed solution, there seems to be an implicit agreement that metaknowledge should be integrated seamlessly into the data mining system. Metalearning focuses on the design and application of learning algorithms to acquire and use metaknowledge to assist machine learning users with the process of model selection. A general framework for this purpose, together with a survey of approaches, is in (Smith-Miles, 2008).

Metalearning is often seen as a way of redefining the space of inductive hypotheses searched by the learning algorithm(s). This issue is related to the idea of ▶search bias, that is, search factors that affect the definition or selection of inductive hypotheses (Mitchell, 1997). In this sense, metalearning studies how to choose the right bias dynamically and thus, differs from base-level learning, where the bias is fixed or user-parameterized. Metalearning can also be viewed as an important feature of self-adaptive systems, that is, learning systems that increase in efficiency through experience (Vilalta & Drissi, 2002).

Structure of the Metalearning System

A metalearning system is essentially composed of two parts. One part is concerned with the acquisition of metaknowledge for machine learning systems. The other part is concerned with the application of metaknowledge to new problems, with the objective of identifying an optimal learning algorithm or technique. The latter part – application of metaknowledge – can be used to help select or adopt suitable machine learning algorithms. So, for instance, if we are dealing with a ▶classification task, metaknowledge can be used to select a suitable ▶classifier for the new problem. Once this has been done, one can train the classifier and apply it to some unclassified sample for the purpose of class prediction.

In the following sections we begin by describing scenarios corresponding to the case when metaknowledge has already been acquired. We then provide an explanation of how this knowledge is acquired.

Employing Metaknowledge to Select Machine Learning Algorithms

The aim of this section is to show that metaknowledge can be useful in many different settings. We start by considering the problem of selecting suitable machine learning algorithms from a given set. The problem can be seen as a search problem. The search space includes the individual machine learning algorithms, and the aim is to identify the best algorithm. This process can be divided into two separate phases (see Fig. 1). In the first phase the aim is to identify a suitable subset of machine learning algorithms based on an input dataset. The selection method used in this process can exploit metaknowledge. This is in general advantageous, as it often leads to better choices. In some work the result of this phase is represented in the form of a ranked subset of machine learning algorithms. The subset of algorithms represents the reduced bias space. The ranking (i.e., ordering of different algorithms) represents the procedural search bias.

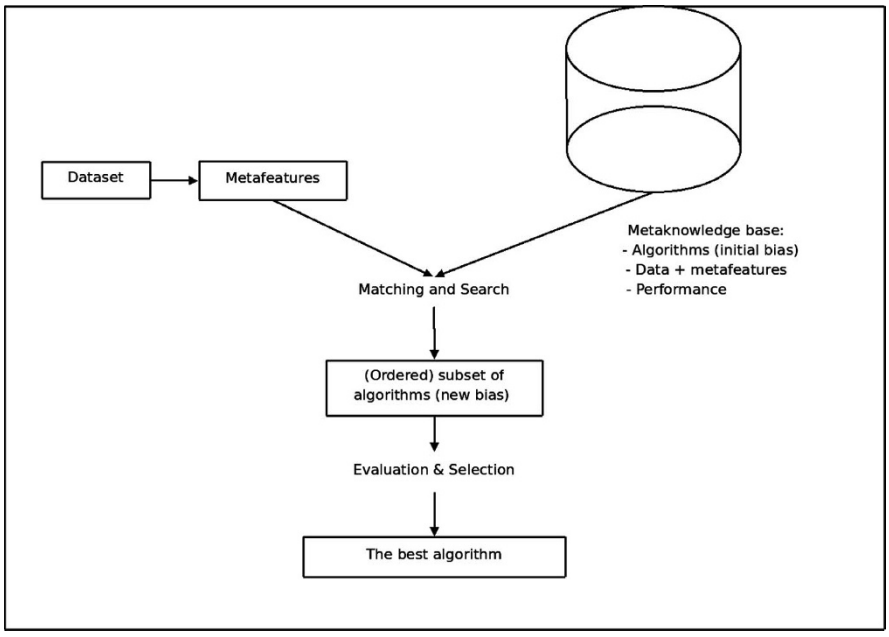
The second phase is used to search through the reduced space. Each option is evaluated using a given performance criteria (e.g., accuracy). Typically, cross-validation will be used to identify the best alternative.

We note that metaknowledge does not completely eliminate the need for the search process, but rather provides a more effective search. The effectiveness of the search depends on the quality of metaknowledge.

How the Subset of Algorithms Is Identified

Let us return to the algorithm selection problem. A metalearning approach to solving this problem relies on dataset characteristics or metafeatures to provide some information that would differentiate the performance of a set of given learning algorithms. These include various types of measures discussed in detail below.

Much previous work in dataset characterization has concentrated on extracting statistical and information-theoretic parameters estimated from the training set.



Metalearning. Figure 1. Selection of machine learning algorithms: Determining the reduced space and selecting the best alternative

Measures include number of classes, number of features, ratio of examples to features, degree of correlation between features and target concept, average class entropy, etc. (Engels & Theusinger, 1998). The disadvantage of this approach is that there is a limit to how much information these features can capture, given that all these measures are uni- or bi-lateral measures only (i.e., they capture relationships between two attributes only or between one attribute and the class).

Another idea is based on what are called *landmarkers;* these are simple and fast learners (Pfahringer, Bensusan & Giraud-Carrier, 2000). The accuracy of these simplified algorithms is used to characterize a dataset and to identify areas where each type of learner can be regarded as an expert. An important class of measures related to landmarkers uses information obtained on simplified versions of the data (e.g., samples). Accuracy results on these samples serve to characterize individual datasets and are referred to as *subsampling landmarks*.

One different class of techniques does not acquire the information in one step, but rather uses a kind of ▶active learning approach. This approach has been used to characterize algorithms by exploiting performance results on samples. The process of obtaining a characterization is divided into several steps. The result of one step affects what is done in the next step. In each step, a decision as to whether the characterization process should be continued is made first. If the answer is positive, the system determines which characteristics should be obtained in the next step (Brazdil et al., 2009).

All the measures discussed above are used to identify a subset of learning algorithms to reduce the search space (Fig. 1). The second phase in the algorithm selection problem can be done using a metalevel system that maps data characteristics to learning algorithms. One particular approach uses the *k*-NN method at the metalevel. The *k*-NN method is used to identify the most similar datasets. For each of these datasets, a ranking of the candidate algorithms is generated based on user-defined performance criteria, such as accuracy and learning time (Nakhaeizadeh & Schnabl, 1997). The rankings obtained are aggregated to generate a final recommended ranking of algorithms.

Acquisition of Metaknowledge

We now address how metaknowledge can be acquired. One possibility is to rely on expert knowledge. Another possibility is to use an automatic procedure. We explore both alternatives briefly below.

One way of representing metaknowledge is in the form of rules that match domain (dataset) characteristics with machine learning algorithms. Such rules can be hand-crafted, taking into account theoretical results, human expertise, and empirical evidence. For example, in decision tree learning, a heuristic rule can be used to switch from univariate tests to linear tests if there is a need to construct nonorthogonal partitions over the input space. This method has serious disadvantages, however. First, the resulting rule set is likely to be incomplete. Second, timely and accurate maintenance of the rule set as new machine learning algorithms become available is problematic. As a result, most research has focused on automatic methods, discussed next.

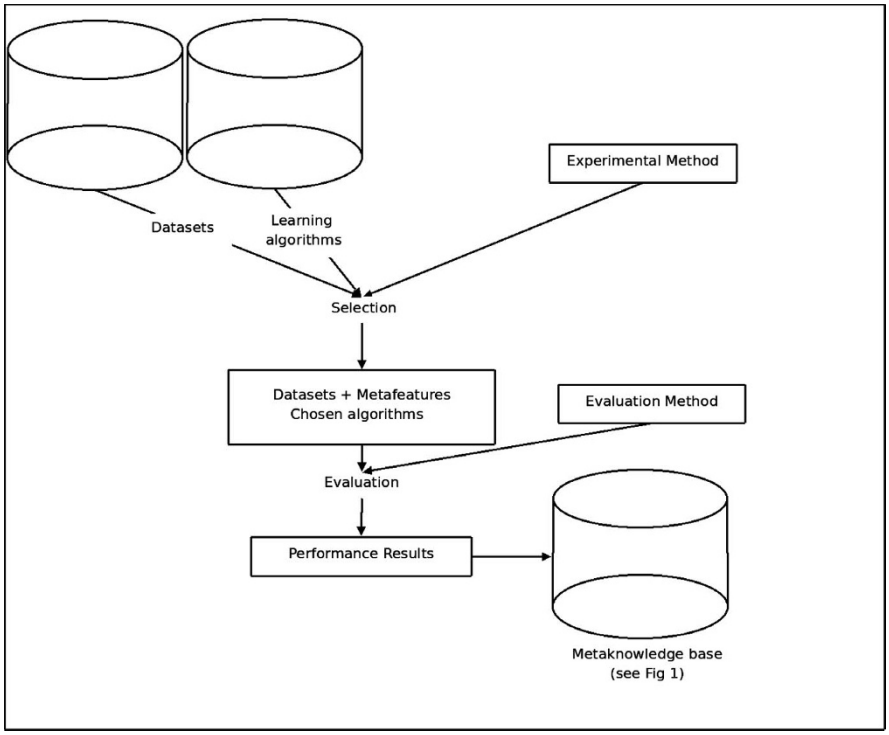
One other way of acquiring metaknowledge relies on automatic experimentation. For this we need a pool of problems (datasets) and a set of machine learning algorithms that we wish to consider. Then we need to define also the experimental method which determines which alternatives we should experiment with and in which order (see Fig. 2 for details).

Suppose that we have a dataset (characterized using certain metafeatures), in combination with certain machine learning algorithms. The combination is assessed using an evaluation method (e.g., cross-validation) to produce performance results. The results, together with the characterization, represent a piece of metadata that is stored in the metaknowledge base. The process is then repeated for other combinations of datasets and algorithms.

In this context it is useful to distinguish between two different types of methods potentially available at the metalevel. One group involves ▶lazy learning methods. These delay the generalization of metadata to the application phase. The other group involves learning algorithms whose aim is to generate a generalization model (e.g., a decision tree or decision rules). This generalization model, applied to the metadatabase, represents in effect the acquired metaknowledge.

Inductive Transfer

As we mentioned before, learning should not be viewed as an isolated task that starts from scratch on every



Metalearning. Figure 2. Acquisition of metadata for the metaknowledge base

new problem. As experience accumulates, the learning mechanism is expected to perform increasingly better. One approach to simulate the accumulation of experience is by transferring metaknowledge across domains or tasks. This process is known as *inductive transfer*. In many cases the goal is not simply to generate explicit metaknowledge, but rather to incorporate it in the given base-level system(s). The resulting base-level system thus becomes a generic solution applicable across domains. More details about this can be found in a separate entry on this topic.

Cross References

►Inductive Transfer

Recommended Reading

Bernstein, A., Provost, F., & Hill, S. (2005). Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification. *IEEE Transactions on Knowledge and Data Engineering, 17*(4), 503–518.

Brazdil, P., Giraud-Carrier, C., Soares, C., & Vilalta, R. (2009). *Metalearning – applications to data mining*. Berlin: Springer.

Brazdil, P., & Henery, R. (1994). Analysis of results. In D. Michie, D. J. Spiegelhalter, & C. C. Taylor (Eds.), *Machine learning, neural and statistical classification*. England: Ellis Horwood.

Engels, R., & Theusinger, C. (1998). Using a data metric for offering preprocessing advice in data-mining applications. In H. Prade (Ed.), *Proceedings of the 13th European conference on artificial intelligence* (pp. 430–434). Chichester, England: Wiley.

Mitchell, T. (1997). *Machine learning*. New York: McGraw Hill.

Nakhaeizadeh, G., & Schnabl, A. (1997). Development of multi-criteria metrics for evaluation of data mining algorithms. In *Proceedings of the 3rd international conference on knowledge discovery and data mining* (pp. 37–42). Newport Beach, CA: AAAI Press.

Pfahringer, B., Bensusan, H., & Giraud-Carrier, C. (2000). Meta-learning by landmarking various learning algorithms. In *Proceedings of the 17th international conference on machine learning* (pp. 743–750).

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers, 15*, 65–118.

Smith-Miles, K. A. (2008). Cross-disciplinary perpectives on meta-learning for algorithm selection. *ACM Computing Surveys, 41*(1), Article No. 6.

Vilalta, R., & Drissi, Y. (2002). A perspective view and survey of metalearning. *Artificial Intelligence Review, 18*(2), 77–95.

Minimum Cuts

►Graph Clustering

Minimum Description Length Principle

Jorma Rissanen
Helsinki Institute of Information Technology, Helsinki, Finland
Tampere University of Technology, Finland
University of London, England

Synonyms

Information theory; MDL; Minimum encoding inference

Definition

The original "general" minimum description length (MDL) principle for estimation of statistical properties in observed data y^n, or the *model* $f(y^n;\theta,k)$, represented by parameters $\theta = \theta_1,\ldots,\theta_k$, can be stated thus,

- "*Find the model with which observed data and the model can be encoded with shortest code length*":

$$\min_{\theta,k}\,[\,\log 1/f(y^n;\theta,k) + L(\theta,k)\,],$$

where $L(\theta,k)$ denotes the code length for the parameters.

The principle is very general and produces a model defined by the estimated parameters. It leaves the selection of $L(\theta,k)$ open, and in complex applications the code length can be calculated by visualizing a coding process. The only requirement is that the data must be decodable.

Motivation and Background

The MDL principle is based on the fact that it is not possible to compress data well without taking advantage of the regular features in them. Hence, estimation and data compression have similar goals although they are not identical. In estimation, we must describe the model explicitly, while the algorithm to compress the data may take advantage of regular features implicitly without isolating them. This means that such an algorithm does not produce any model that could be used for machine learning.

We describe here a new sharper *Complete MDL* principle, which corrects the shortcomings of the old principle. It also separates estimation from data compression and delivers an explicit model. The objective in this entry is to show that the complete MDL principle is not only intuitively appealing but it plays a fundamental role in all estimations, and in all inductive inferences for that matter, since any meaningful inference about data must be based on a good model of it. In fact, we argue that there cannot be a rational comprehensive theory of estimation unless it is founded on the MDL principle or some of its equivalent forms.

Theory

We begin by outlining the problem of model building and estimation. The objective is to fit parametric models of type $f(y^n|\mathbf{x}^n, \theta)$ to data $y^n = y_1, \ldots, y_n$, given explanatory variables $\mathbf{x}^n = \mathbf{x}_1, \ldots, \mathbf{x}_n$, where $\theta = \theta_1, \ldots, \theta_k$ are real-valued parameters. To simplify the notations we drop the explanatory variables and consider classes of models $\mathcal{M}_k = \{f(y^n;\theta)\}$. For fixed number k of parameters the fitting is done by some *estimator* function

$$\bar{\theta}(\cdot) : y^n \mapsto \bar{\theta}(y^n)$$

taking data to parameter values, which in turn pick out estimated models of data. For simplicity we discuss first the estimation of the real-valued parameters. We do not assume the existence of a special "true" model defined by a parameter θ^*, which raises the problem of how to assess the goodness of the estimators and the estimated models. Clearly, it cannot be done by any distance measure between the estimate $\bar{\theta}(y^n)$ and θ^*. We think that the only way the assessment can be done in general without narrow and special criteria is in terms of the probability which the estimator $\bar{\theta}(\cdot)$ assigns to the observed data. A large probability means a good fit while a small probability means a bad fit. How do we calculate this probability? Importantly, notice that it cannot be the number $f(y^n;\bar{\theta}(y^n))$, bearing the mystical name "likelihood," because its integral over all data y^n is not unity. However, by normalization we get a valid yardstick for the goodness measure

$$\bar{f}(y^n;k) = \frac{f(y^n;\bar{\theta}(y^n),k)}{\bar{C}_k}$$

$$\bar{C}_k = \int f(y^n;\bar{\theta}(y^n),k)dy^n,$$

where we now show the number of parameters k. When even the number of parameters is to be estimated, the yardstick is as follows

$$\bar{f}(y^n) = \bar{f}(y^n;\bar{k}(y^n))/\bar{C} \quad (1)$$

$$\bar{C} = \sum_k \int_{\bar{k}(y^n)=k} \bar{f}(y^n;k)dy^n, \quad (2)$$

where $\bar{k}(y^n)$ denotes an estimator for k.

Optimal Yardstick

We view estimation as analogous to measuring a physical property like weight or mass of an object: The object here is the observed data and the property is the model in a selected class defined by the parameters, while the probability an estimated model assigns to the data corresponds to the accuracy.

We need a yardstick as the instrument like the scale with which the measuring is done. It will be defined by a special estimator and the distribution it defines. Clearly, the yardstick must not depend on the data set whose property we want to measure no more than the scale for weighing an object must not depend on the object. The requirement then is that it should be determined by the model class. We also want a yardstick that assigns a large probability to the data, or, equivalently, a small negative logarithm of the probability, which can be interpreted as code length. However, there is the fundamental difficulty that no distribution $\bar{f}(y^n;k)$ exists which assigns the largest probability to all data. Quite remarkably, there is a unique yardstick that satisfies the two requirements, repeated here:

1. $\bar{f}(\cdot;k)$ to be determined by the model class $\mathcal{M}_k$
2. Minimal code length $\log 1/\bar{f}(y^n;k)$ for *all* data y^n

and, similarly, when even the number of parameters is to be estimated.

The unique yardstick when the real-valued parameters are estimated is defined by the ML (maximum Likelihood) estimator, $\hat{\theta}(y^n)$, which maximizes the probability the model assigns to data, or $\max_\theta f(y^n;\theta,k)$:

$$\hat{f}(y^n;k) = \frac{f(y^n;\hat{\theta}(y^n),k)}{\hat{C}_k} \quad (3)$$

$$\hat{C}_k = \int f(y^n;\hat{\theta}(y^n),k)dy^n$$

$$= \int d\hat{\theta} \int_{\hat{\theta}(y^n)=\hat{\theta}} f(y^n;\hat{\theta},k)dy^n. \quad (4)$$

The proof of that there is a unique distribution $\bar{f}(y^n;k) = \hat{f}(y^n;k)$ satisfying the two requirements amounts to noticing that the ratio $\bar{f}(y^n;k)$ cannot be maximum unless the numerator is maximized. Notice that the famous maximized likelihood, the numerator of $\hat{f}(y^n;k)$, in itself means nothing.

The unique yardstick when even the number of parameters is estimated is

$$\hat{f}(y^n) = \frac{\max_k f(y^n;\hat{\theta}(y^n),k)/\hat{C}_k}{\hat{C}} \tag{5}$$

$$\hat{C} = \sum_k \int_{\hat{k}(y^n)=k} \hat{f}(y^n;k)dy^n, \tag{6}$$

where $\hat{k}(y^n)$, or the maximizing k, is not the maximum likelihood estimator.

The evaluation of these yardsticks on an observed data string y^n gives the MDL criterion. The calculation of the normalizing coefficients is the main problem. It can be evaluated most easily for finite alphabets. Asymptotically the optimal estimation criterion amounts to this

$$\min_k \left[\log 1/f(y^n;\hat{\theta}(y^n),k) + \frac{k}{2}\log\frac{n}{2\pi} + \log\int |J(\theta)|^{1/2}d\theta\right],$$

where

$$J(\theta) = \lim n^{-1}E\left\{\frac{\partial^2 \log 1/f(y^n;\theta,k)}{\partial\theta_i\partial\theta_j}\right\}.$$

is the Fisher information matrix. Hence, this term is a positive constant and can be ignored for large amounts of data.

Cross References

►Minimum Message Length

Recommended Reading

Grünwald, P. D. (2007). *The minimum description length principle* (703 pp.). Cambridge/London: The MIT Press.

Rissanen, J. (2007). *Information and complexity in statistical modeling* (142 pp.). Springer: New York.

Rissanen, J. (September 2009). Optimal estimation. *IEEE Information Theory Society Newsletter*, 59(3).

Minimum Encoding Inference

►Minimum Description Length Principle
►Minimum Message Length

Minimum Message Length

Rohan A. Baxter
Australian Taxation Office, ACT, Australia

Synonyms

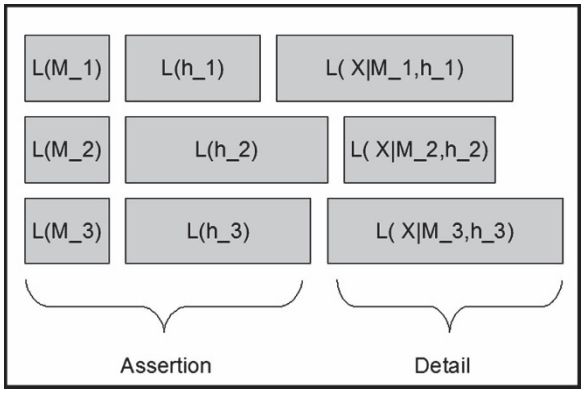
Minimum encoding inference

Definition

Minimum message length (MML) is a theory of ►inductive inference whereby the preferred model is the one minimizing the expected message length required to explain the data with the prior information.

Given data, represented in a finite binary string, E, is an "►explanation" of the data which is a two-part ►message or binary string encoding the data to be sent between a sender and receiver. The first part of the message (the "►assertion") states a hypothesis, model, or theory about the source of the data. The second part (the "►detail") states those aspects of E which cannot be deduced from this assertion and prior knowledge. The sender and receiver are assumed to have agreed on the prior knowledge, the assertion code, and the detail code before the message is constructed and sent. The shared prior knowledge captures their belief about the data prior to seeing the data and is needed to provide probabilities or, equivalently, optimum codes, for the set of models. The assertion and detail codes can be equivalently considered to be the shared language for describing models (for the assertion code) and for describing data (for the detail code).

Out of all possible models which might be advanced about the data, MML considers the best inference as that model which leads to the shortest explanation. The length of the explanation can be calculated using ►Shannon's information, $L(E) = -\log(P(E))$, where $L(E)$ is the length of the shortest string encoding an



Minimum Message Length. Figure 1. A view of model selection by minimum message length (MML). The data is coded assuming a model and parameters in the assertion. The model and parameters are coded in the assertion. As shown here, often different models have same probability, while the code lengths for model parameters and data detail differ between the models

event, E, and $P()$ is the probability of a message containing E.

To compare models, we calculate the explanation length for each and prefer the one with shortest explanation length. Figure 1 shows three models being evaluated and the different lengths of the assertion and details for each. Model 2 is preferred as it has the MML.

Motivation and Background

The original motivation for MML inductive inference is the idea that the best explanation of the facts is the shortest (Wallace & Boulton, 1968). By inductive inference, we mean the selection of a best model of truth. This goal is distinct from a best model for prediction of future data or for choosing a model for making the most beneficial decisions. In the field of Machine Learning, greater focus has been on models for prediction and decision, but inferences of the best models of truth have an important separate application.

For discrete models, MML looks like Bayesian model selection since choosing H to minimize the explanation length of data X:

$$-\log P(H) - \log P(X|H) = -\log(P(H)P(X|H),$$

is often, but not always, as discussed below, equivalent to choosing H to maximize the probability

$$P(H|X):$$
$$P(H|X) = \frac{P(H)P(X|H)}{P(X)},$$

where $P(X)$ is a constant for a given detail code.

For models with real-valued parameters, the equivalence between MML and Bayesian model selection always breaks down (Wallace, 2005, p. 117). Stating the $P(H)$ in a message requires real-valued parameters in H to be stated to a limited precision. The MML coding approach replaces a continuum of possible hypotheses with a discrete subset of values, and assigns a nonzero prior probability to each discrete theory. The discrete subsets are chosen to optimize the expected message length given the prior knowledge assumptions.

For models with only discrete-valued parameters, the equivalence between MML and Bayesian model selection may break down if the discrete values chosen involve the merging of values in the assumed prior distribution, $P(H)$ (Wallace, 2005, p. 156). This may occur with a small dataset if the data is insufficient to justify a codebook distinguishing individual members of H.

Other than a discretized hypothesis space, MML shares many properties of Bayesian learning such as sufficiency, avoidance of overfitting, and consistency (Wallace, 2005). One difference arising from the discretized hypothesis space is that MML allows inductive inference to be invariant under arbitrary monotonic transformations of parameter spaces. The Bayesian learning options for model choice such as the maximum a posteriori (MAP) estimate are not invariant under such transformations. Other theoretical benefits include consistency and guarantees against overfitting.

Message lengths of an explanation can be based on the theory of algorithmic complexity (AC) (Wallace & Dowe, 1999), instead of Shannon's information. The AC of a string with respect to a Universal Turing Machine, T, can be related to Shannon's information by

regarding T as defining a probability distribution over binary strings, $P(S)$, such that:

$$P_{T(S)} = 2^{-\mathrm{AC}(S)} \quad \forall S.$$

The connection with AC has some appeal for applications involving data that are not random in a probabilistic sense, such as function approximation where data seems to be from a deterministic source. In these cases, after fitting a model, the data residuals can be encoded using AC randomness, since the probabilistic sense of randomness does not apply (Wallace, 2005, p. 275).

Theory

Strict MML (SMML) estimators refer to the estimator functions which exactly minimize the expected message length (Wallace & Boulton, 1975). Most practical MML estimators are not strict and are discussed in a separate section on Approximations.

An SMML estimator requires (Dowe, Gardner, & Oppy, 2007):

- X, a data space, and a set of observations from the dataspace, $\{x_i : i \in N\}$
- $p(x|h)$, a conditional probability function over data given a model, h
- H is a model space. For example, H can be a simple continuum of known dimension k
- $P(h)$: a prior probability density on the parameter space $H : \int_H P(h)\,\mathrm{d}h = 1$

X, H, and the functions $\mathrm{P}(h), p(x|h)$ are assumed to be known a priori by both sender and receiver of the explanation message. Both sender and receiver agree on a code for X, using knowledge of X, H, $p(h)$, and $f(x|h)$ only.

The marginal prior probability of the data x follows from the assumed background knowledge:

$$r(x) = \int_H p(x|h)P(h)\,\mathrm{d}h.$$

The SMML estimator is a function $m : X \to H : m(x) = h$, which names the model to be selected.

The assertion, being a finite string, can name at most a countable subset of H. Call the subset $H^* = \{h_j : j = 1, 2, 3, \ldots\}$. The choice of H^* implies a coding distribution over $H^* : f(h_j) = q_j > 0 : j = 1, 2, 3, \ldots$ with $\sum_j q_j = 1$. So choice of H^* and q_j lead to a message length:

$$-\log q_j - \log p(x|h_j).$$

The sender, given x, will choose an h to make the explanation short. This choice is described by an estimator function: $m(x) : X \to H$ so that the length of the explanation is:

$$I_1(x) = -\log q(m(x)) - \log p(x|m(x)),$$

and the expected length is (Wallace, 2005, p. 155)

$$I_1 = -\sum_{x \in X} r(x)[\log q(m(x)) + \log p(x_i|m(x_i))].$$

Consider how to choose H^* and coding distribution q_j to minimize I_1. This will give the shortest explanation on average, prior to the sender seeing the actual data.

Define $t_j = \{x : m(x) = h_j\}$, so that t_j is the set of data which results in assertion h_j being used in the explanation. I_1 can now be written as two terms:

$$I_1 = -\sum_{h_j \in H_{\text{start}}} \left(\sum_{x_i \in t_j} r_i \right) \log q_j - \sum_{h_j \in H_{\text{start}}} \sum_{x_i \in t_j} r_i \log p(x_i|h_j).$$

The first term of I_1 is minimized by choosing:

$$q_j = \sum_{x_i \in t_j} r_j.$$

So the coding probability assigned to estimate h_j is the sum of the marginal probabilities of the data values resulting in h_j. It is the probability that estimate h_j will be used in the explanation based on the assumptions made.

The second term of I_1 is the average of the log likelihood over the data values used in h_j.

Example with Binomial Distribution

This section describes the SMML estimator for the binomial distribution. For this problem with 100 independent trials giving success or failure, we have $p(x|p) = p^n(1-p)^{100} - s, h(p) = 1$, where s is the observed number of successes and p is the unknown probability of success.

Minimum Message Length. Table 1 A strict MML (SMML) estimator for binomial distribution (Farr & Wallace, 2002; Wallace, 2005, p. 159)

j	s	p_j
1	0	0
2	1–6	0.035
3	7–17	0.12
4	18–32	0.25
5	33–49	0.41
6	50–66	0.58
7	67–81	0.74
8	82–93	0.875
9	94–99	0.965
10	100	1

We have an SMML estimator minimizing I_1 in Table 1. I_1 has 52.068 nits. Note that the partition p_j in Table 1 is not unique due to asymmetry in having 10 partitions of 101 success counts. Note the difference between the SMML estimate, p_j, and the MAP estimate $s/100$ in this case. For example of 50 observed successes, the MAP estimate is 0.5 while SMML estimate is 0.58. With 49 successes, the SMML estimate jumps to 0.41, so it is very discrete. The SMML estimate spacings are consistent with the expected error and so the MAP estimates are arguably overly precise and smooth. This is less than 0.2 nits more than the optimal one-part code based on the marginal probability of the data $-\log r(x)$.

Approximations

SMML estimators are hard to find in practice and various approximations of SMML estimators have been suggested. We focus on the quadratic approximation here, often called the MML estimator or MML87 (Wallace & Freeman 1987). Other useful approximations have been developed and are described in Wallace, (2005). The use of approximations in applications requires careful checking of the assumptions made by the approximation (such as various regularity conditions) to ensure that the desirable theoretical properties of MML inductive inference still apply.

$$I_1(x) \approx -\log \frac{f(h')}{\sqrt{\frac{F(h')}{12}}} + [-\log p(x|h')] + \frac{0.5F(h',x)}{F(h')},$$

where $F(h)$ is the Fisher Information:

$$F(h') = -E\frac{\partial^2}{(\partial h')^2}\log p(x|h')$$

$$= -\sum_{x\in X} p(x|h')\frac{\partial^2}{(\partial h')^2}\log p(x|h').$$

The assumptions are (Wallace, 2005; Wallace & Freeman, 1987):

- $f(x|h)$ is approximately quadratic on theta near its maximum
- H has a locally Euclidean metric
- Fisher information is defined everywhere in H
- $f(h)$and $F(h)$ vary little over theta of order $1/\sqrt{F(h)}$

A further approximation has the third term simplify to 0.5 only (Wallace, 2005, p. 226) which assumes $F(h,x) \approx F(h)$.

The MML estimator is a discretized MAP estimator with the prior $P(h)$ being discretized as:

$$f(h') \approx \frac{P(h')}{\sqrt{F(h')}}.$$

In practice, note that the Fisher Information may be difficult to evaluate. Various approximations have been made for the Fisher Information where appropriate for particular applications.

Applications

MML estimators have been developed for various probability distributions such as binomial, multinomial, and Poisson. MML estimators have also been developed for probability densities such as Normal, von-Mises, and Student's t (Wallace, 2005). These estimators and associated coding schemes are then useful components for addressing more complex model selection problems in Machine Learning.

There have been many applications of MML estimators to model spaces from Machine Learning (Allison, 2009; O'Donnell, Allison, & Korb, 2006; Wallace, 2005). We will now briefly note MML applications for mixture models, regular grammars, decision trees, and causal nets. MML estimators have also been developed for multiple ▶linear regression (Wallace, 2005), polynomial regression (Wallace, 2005), ▶neural networks (Allison, 2009), ARMA time series, Hidden Markov Models (Edgoose & Allison, 1999), sequence alignment (Allison, 2009), phylogenetic trees (Allison, 2009), factor analysis (Wallace, 2005), cut-point estimation (Wallace, 2005), and image segmentation.

Model-Based Clustering or Mixture Models

Clustering was the first MML application from Wallace and Boulton's 1968 paper (Wallace & Boulton, 1968). Some changes to the coding scheme have occurred over the decades. A key development was the switch from definite assignment of classes to things to probabilistic assignment in the 1980s. The MML model selection and a particularly efficient search involving dynamic splitting and merging of clusters was implemented in a FORTRAN program called Snob (since it discriminated between things).

The assertion code consists of:

1. The number of classes
2. For each class
   (a) The population proportion
   (b) Parameters of the statistical distribution for each attribute (or an insignificant flag)

The detail code consists of, for each datum the class to which it belongs and attribute values assuming the distribution parameters of the class. Bits-back coding is used to partially or probabilistically assign a class to each datum. This efficiency is needed to get consistent estimates.

Probabilistic Finite State Machines

Probabilistic finite state machines (PFSM) can represent probabilistic regular grammars (Wallace, 2005). A simple assertion code for the discrete finite state machines (FSM) structure, as developed by Wallace and Georgeff, is as follows:

- Provide number of states, S, using a prior $P(S)$
- For each state, code the number of arcs leaving the state, $\log(K+1)$ where $K+1$ is maximum number of arcs possible
- Code the symbols labeling the arcs, $\log \binom{K+1}{C\ a_s}$
- For each arc, code the destination state, $a_s \log S$

The number of all states other than state 1 is arbitrary, so the code permits $(S-1)!$, equal length, different descriptions of the same FSM. This inefficiency can be adjusted for by subtracting $\log(S-1)!$

A candidate detail code used to code the sentences is an incremental code where each transition from state to state is coded incrementally, using $\log n_{sk} + 1/\nu_s + a_s$, where n_{sk} is the number of times this arc has already been followed and ν_s is the number of times the state has already been left.

This application illustrates some general issues about assertion codes for discrete structures:

1. There can be choices about what to include in the assertion code. For example, the transition probabilities are not part of the assertion code above, but could be included, with adjustments, in an alternative design (Wallace, 2005).
2. Simple approaches with interpretable priors may be desirable even if using non-optimal codes. The assumptions made should be validated. For example, arcs between states in FSMs are usually relatively sparse (a_s = S) so a uniform distribution is not a sensible prior here.
3. Redundancy comes from being able to code equivalent models with different descriptions. For some model spaces, determining equivalence is either not possible or very expensive computationally.
4. Redundancy can come from the code allowing description of models that cannot arise. For example, the example assertion code could describe a FSM with states with no arcs.
5. Exhaustive search of model space can only be done for small FSMs. For larger applications, the performance of the MML model selection is conflated with performance of the necessary search space heuristics. This issue also occurs with decision trees, causal nets, etc.

In a particular application, it may be appropriate to trade-off redundancy with interpretability in assertion code design.

Decision Trees

Assertion codes for Decision trees and graphs have been developed (Wallace, 2005; Wallace & Patrick, 1993). An assertion describes the structure of the tree, while the detail code describes the target labels. The number of attributes, the arity of each attribute, an agreed attribute order, and probability that a node is a leaf or split node are assumed known by sender and receiver. Like the PFSM transition probabilities, the leaf class distributions are not explicitly included in the decision tree model (a point of distinction from Bayesian tree approaches).

An assertion code can be constructed by performing a prefix traversal of the tree describing each node. Describing a node requires –log_2 P_L if it is a leaf and –log_2 P_s if it is a split node. If it is a split node, the attribute that it splits on must be specified, requiring log_2 (number of available attributes). If it is a leaf node, the data distribution model should be specified, for example, the parameters of a binomial distribution if the data consists of two classes.

Causal Nets

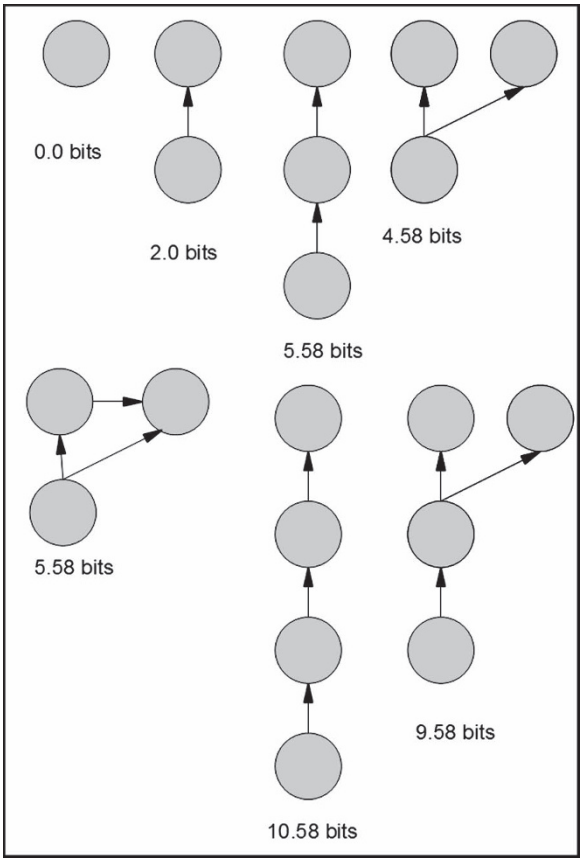
(Dai, Korb, Wallace, & Wu, 1997; Neil, Wallace, & Korb, 1999; O'Donnell et al., 2006)
The assertion code has two parts.
First part: DAG:

1. Specify an ordering of variables, $\log N!$
2. Specify which of M_a possible arcs are present, $\log(N(N-1)/2)$ bits on assumption probability an arc is present is 0.5

Second part: Parameters:

1. For each variable, state form of conditional distribution, then parameters of the distribution. Then encode all N values of v_j according to the distribution (Fig. 2)

Note that the assertion code is intermixed with the detail code for each variable (Wallace, 2005). Further adjustments are made to deal with grouping of causal nets with various equivalences or near-equivalences. This requires a further approximation because no attempt is made to code the best representative causal net from the group of causal nets described.



Minimum Message Length. Figure 2. Assertion code lengths for different DAGS using the example coding scheme

Future Directions

There seems potential for further development of feasible approximations that maintain the key SMML properties. Crossover of exciting new developments in coding theory may also help with development of MML estimators. Examples include stochastic encoding such as bits-back coding, discovered by Wallace (1990) and since expanded to many new application areas showing connections between MML with variational learning and ensemble learning (Honkela & Valpola,

2004). Another area is the relationship between optimum hypothesis discretization and indices of resolvability and rate-distortion optimization (Lanterman, 2001).

MML estimators will continue to be developed for the new model spaces that arise in Machine Learning. MML relevance seems assured because with complex models, such as social networks, the best model is the useful outcome, rather than a prediction or posterior distribution of networks.

Open source software using MML estimators for difference machine learning models is available (MML software).

Cross References

►Bayesian Methods
►Inductive Inference
►Minimum Description Length

Recommended Reading

Allison, L. (2009). 27/1/2010 MML website, http://www.allisons.org/ll/MML/

Dai, H., Korb, K. B., Wallace, C. S., & Wu, X. (1997). A study of causal discovery with weak links and small samples. In *Proceedings of the fifteenth international joint conference on artificial intelligence* (pp. 1304–1309). San Francisco: Morgan Kaufman.

Dowe, D. L., Gardner, S. B., & Oppy, G. (2007). Bayes not bust!: Why simplicity is no problem for Bayesians. *The British Journal for the Philosophy of Science, 58*, 709–754.

Edgoose, T., & Allison, L. (1999). MML Markov classification of sequential data. *Statistics and Computing, 9*(4), 269–278.

Farr, G. E, & Wallace, C. S. (2002). The complexity of strict minimum message length inference. *The Computer Journal, 45*(3), 285–292.

Honkela, A., & Valpola, H. (2004). Variational learning and bits-back coding: An information-theoretic view to Bayesian learning. *IEEE Transactions on Neural Networks, 15*(4), 800–810.

Lanterman, A. D. (2001). Schwarz, Wallace and Rissanen: Intertwining themes in theories of model selection. *International Statistical Review, 69*(2), 185–212.

MMLsoftware: www.datamining.monash.edu.au/software, 27/1/2010 http://allisons.org/ll/Images/People/Wallace/FactorSnob/

Neil, J. R., Wallace, C. S., & Korb, K. B. (1999). Learning Bayesian networks with restricted interactions. In K. B. Laskey & H. Prade (Eds.), *Proceedings of the fifteenth conference of uncertainty in artificial intelligence (UAI-99)* (pp. 486–493). San Francisco: Morgan Kaufmann.

O'Donnell, R., Allison, L., & Korb, K. (2006) *Learning hybrid Bayesian networks by MML. Lecture notes in computer science: AI 2006 – advances in artificial intelligence* (Vol. 4304, pp. 192–203). Berlin: Springer.

Wallace, C. S. (1990). Classification by minimum-message length inference. In S. G. Akl, et al. (Eds.), *Advances in computing and information-ICCI 1990, No. 468 in Lecture notes in computer science*. Berlin: Springer.

Wallace, C. S. (2005). *Statistical and inductive inference by MML: Information sciences and statistics*. Berlin: Springer.

Wallace, C. S., & Boulton, D. M. (1968). An information measure for classification. *Computer Journal, 11*, 185–194.

Wallace, C. S., & Boulton, D. M. (1975). An information measure for single-link classification. *The Computer Journal, 18*(3), 236–238.

Wallace, C. S., & Dowe, D. L. (1999). Minimum message length and Kolmogorov complexity. *Computer Journal, 42*(4), 330–337.

Wallace, C. S., & Freeman, P. R. (1987). Estimation and inference by compact coding. *Journal of the Royal Statistical Society (Series B), 49*, 240–252.

Wallace, C. S., & Patrick, J. D. (1993). Coding decision trees. *Machine Learning, 11*, 7–22.

Missing Attribute Values

Ivan Bruha
McMaster University, Hamilton, ON, Canada

Synonyms

Missing values; Unknown attribute values; Unknown values

Definition

When inducing ►decision trees or ►decision rules from real-world data, many different aspects must be taken into account. One important aspect, in particular, is the processing of *missing* (*unknown*) ►attribute values. In machine learning (ML), instances (objects, observations) are usually represented by a list of attribute values; such a list commonly has a fixed length (i.e., a fixed number of attributes).

The topic of missing attribute values has been analyzed in the field of ML in many papers (Brazdil & Bruha, 1992; Bruha and Franek, 1996; Karmaker & Kwer, 2005; Long & Zhang, 2004; Quinlan, 1986, 1989). Grzymala-Basse (2003) and Li and Cercone (2006) discuss the treatment of missing attribute values using the rough set strategies.

There are a few directions in which missing (unknown) attribute values as well as the corresponding routines for their processing may be studied and designed. First, the *source of "unknownness"* should be investigated; there are several such sources (Kononenko, 1992):

- A value is *missing* because it was forgotten or lost
- A certain attribute is *not applicable* for a given instance (e.g., it does not exist for a given observation)
- An attribute value is *irrelevant* in a given context
- For a given observation, the designer of a training database does not care about the value of a certain attribute (the so-called *dont-care* value)

The first source may represent a random case, while the remaining ones are of structural character.

Moreover, it is important to define formulas for *matching instances* (examples) containing missing attribute values with decision trees and decision rules as different matching routines vary in this respect.

Strategies for Missing Value Processing

The aim of this section is to survey the well-known strategies for the processing of missing attribute values. Quinlan (1989) surveys and investigates quite a few techniques for processing unknown attribute values processing for the TDIDT family. This chapter first introduces the seven strategies that are applied in many ML algorithms. It then discusses particular strategies for the four paradigms: Top Down Induction Decision Trees (TDIDT), (also known as the decision tree paradigm, or divide-and-conquer), covering paradigm (also known as the decision rules paradigm), Naive Bayes, and induction of ▶association rules. The conclusion compares the above strategies and then portrays possible directions in combining these strategies into a more robust system.

To deal with real-world situations, it is necessary to process incomplete data – i.e., data with missing (unknown) attribute values. Here we introduce the seven strategies (routines) for processing missing-attribute-values. They differ in the style of the solution of their matching formulae. There are the following natural ways of dealing with unknown attribute values:

1. Ignore the example (object, observation) with missing values: strategy *Ignore* (*I*)
2. Consider the missing (unknown) value as an additional regular value for a given attribute: strategy *Unknown* (*U*) or
3. Substitute the missing (unknown) value for matching purposes by a suitable value which is either
   - The most common value: strategy *Common* (*C*)
   - A proportional fraction: strategy *Fraction* (*F*)
   - Any value: strategy *Anyvalue* (*A*)
   - Random value: strategy *Random* (*Ran*)
   - A value determined by a ML approach: strategy *Meta-Fill-In* (*M*) of the known values of the attribute that occur in the training set

Dealing with missing attribute values is in fact determined by matching a selector (see the corresponding definitions below) with an instance. A matching procedure of a selector with a fully specified instance returns the uniform solution: the instance either matches or not. Dilemmas arise when a partially defined instance is to be matched.

We now informally introduce a couple of definitions. An inductive algorithm generates a knowledge base (decision tree or a set of decision rules) from a *training set* of K training examples, each accompanied by its desired ▶class $C_r, r = 1, \ldots, R$. Examples are formally represented by N ▶attributes, which are either discrete (symbolic) or numerical (continuous). A discrete attribute $A_n, n = 1 \ldots, N$, comprises $J(n)$ distinct values $V_1, \ldots, V_{J(n)}$. A numerical attribute may attain any value from a continuous interval. The symbolic/logical ML algorithms usually process the numerical attributes by ▶discretization/fuzzification procedures, either on-line or off-line; see e.g., Bruha and Berka (2000).

An example (object, observation) can thus be expressed as an N-tuple $\mathbf{x} = [x_1, \ldots, x_N]$, involving N attribute values. A *selector* S_n is defined as an attribute-value pair of the form $x_n = V_j$, where V_j is the jth value of the attribute A_n (or the jth interval of a numerical attribute A_n).

To process missing values, we should know in advance (for $r = 1, \ldots, R, n = 1, \ldots, N, j = 1, \ldots, J(n)$):

- The overall *absolute* frequencies $F_{n,j}$ that express the number of examples exhibiting the value V_j for each attribute A_n

- The *class-sensitive absolute* frequencies $F_{r,n,j}$ that express the number of examples of the class C_r exhibiting the value V_j for each attribute A_n
- The *overall relative* frequencies $f_{n,j}$ of all known values V_j for each attribute A_n
- The *class-sensitive relative* frequencies $f_{r,n,j}$ of all known values V_j for each attribute A_n and for a given class C_r

The underlying idea for learning relies on the class distribution; i.e., the class-sensitive frequencies (overall and class-sensitive frequencies) are utilized. As soon as we substitute a missing value by a suitable one, we take the desired class of the example into consideration in order not to increase the noise in the data set. On the other hand, the overall frequencies are applied within classification.

We can now define the matching of an example **x** with a selector S_n by the so-called *matching ratio* $= 0$ if $x_n \neq V_j$

$$\mu(\mathbf{x}, S_n)\{= 1 \text{ if } x_n = V_j \quad (1)$$

$$\in [0;1] \text{ if } x_n \text{ is unknown (missing)}$$

A particular value of the matching ratio is determined by the selected routine (strategy) for missing value processing.

(I) Strategy Ignore: Ignore Missing Values: This strategy simply ignores examples (instances) with at least one missing attribute value before learning. Hence, no dilemma arises when determining matching ratios within learning. However, this approach does not contribute to any enhancement of processing of noisy or partly specified data.

As for classification, a missing value does not match any regular (known) value of a selector. Thus, a selector's matching ratio is equal to 0 for any missing value. Consequently, only a path of nodes in a decision tree or a decision rule that tests only the regular values during classification may succeed. If there is no such path of nodes in a decision tree or such a rule has not been found, then the default principle is applied; i.e., the instance with missing value(s) is classified as belonging to the majority class.

(U) Strategy Unknown: Unknown Value as a Regular One: An unknown (missing) value is considered as an additional attribute value. Hence, the number of values is increased by one for each attribute that depicts an unknown value in the training set. The matching ratio of a selector comprising the test of the selector S_n and an instance with the nth attribute missing is equal to 1 if this test (selector) is of the form $x_n =?$ where "?," represents the missing (unknown) value.

Note that selectors corresponding to the numerical (continuous) attributes are formed by tests $x_n \in V_j$ (where V_j is a numerical interval) or $x_n =?$.

(C) Strategy Common: The Most Common Value: This routine needs the class-sensitive absolute frequencies $F_{r,n,j}$ to be known before the actual learning process, and the overall frequencies $F_{n,j}$ before the classification. A missing value of a discrete attribute A_n of an example belonging to the class C_r is replaced by the *class-sensitive common* value, which maximizes the Laplacian formula $\frac{F_{r,n,j}+1}{F_{n,j}+R}$ over j for the given r and n. If the maximum is reached for more than one value of A_n, then the value V_j with the greatest frequency $F_{r,n,j}$ is selected as the common value.

A missing value within the classification is replaced by the *overall common* value, which maximizes $F_{n,j}$ over the subscript j. Consequently, the matching ratio yields 0 or 1, as every missing value is substituted by a concrete, known value.

The Laplacian formula utilized within the learning phase prefers those attribute values that are more predictive for a given class, contrary to the conventional "maximum frequency" scheme. For instance, let an attribute have two values: the value V_1 with the absolute frequencies [4, 2] for the classes C_1 and C_2, and the value V_2 with frequencies [3, 0] for these two classes. Then, when looking for the most common value of this attribute for the class C_1, the maximum frequency chooses the value V_1 as the most common value, whereas the Laplacian formula prefers the value V_2 as the more predictive for the class C_1.

(F) Strategy Fraction: Split into Proportional Fractions:

- Learning phase

The learning phase requires that the relative frequencies $f_{r,n,j}$ above the entire training set be known. Each example **x** of class C_r with a missing value of a discrete attribute A_n is substituted by a collection of examples

before the actual learning phase, as follows: the missing value of A_n is replaced by all known values V_j of A_n and C_r. The weight of each split example (with the value V_j) is

$$w_j = w(\mathbf{x}) * f_{r,n,j}, j = 1, \ldots, J(n)$$

where $w(\mathbf{x})$ is the weight of the original example $\mathbf{x}$. The weight is assigned by the designer of the training set and represents the designer's subjective judgment of the importance of that particular example within the entire training set. The matching ratio of the split examples is accomplished by (1) in a standard way.

If a training example involves more missing attribute values, then the above splitting is done for each missing value. Thus, the matching ratio may rapidly decrease. Therefore, this strategy, *Fraction,* should involve a methodology to avoid explosion of examples, so that only a predefined number of split examples with the largest weights is used for replacement of the original example.

- Classification phase

The routine *Fraction* works for each paradigm in a different way. In case of a decision tree, the example with a missing value for a given attribute A_n is split along all branches, with the weights equal to the overall relative frequencies $f_{n,j}$.

As for the decision rules, the matching ratio for a selector $x_n = V_j$ is defined by (1) as $\mu = f_{n,j}$ for a missing value of A_n. An instance with a missing value is tested with the conditions of all the rules, and is attached to the rule whose condition yields the maximum matching ratio – i.e., it is assigned to the class of this rule.

(A) Strategy Anyvalue: Any Value Matches: A missing value matches any existing attribute value, both in learning and classification. Therefore, a matching ratio μ of any selector is equal to 1 for any missing value.

It should be noticed that there is no uniform scheme in machine learning for processing the "any-value." In some systems, an example with a missing value for attribute A_n is replaced by $J(n)$ examples in which the missing value is in turn substituted by each regular value $V_j, j = 1, \ldots, J(n)$. In other systems, the missing "any-value" is substituted by any first attribute value involved in a newly generated rule when covered examples are being removed from the training set; see Bruha and Franek (1996) for details.

(Ran) Strategy Random: Substitute by Random Value A missing value of an attribute A_n is substituted by a randomly selected value from the set of its values $V_j, j = 1, \ldots, J(n)$. In case of the numerical attributes, the process used in the routine *Common* is first applied, i.e., the entire numerical range is partitioned into a pre-specified number of equal-length intervals. A missing value of the numerical attribute is then substituted by the mean value of a randomly selected interval.

At least two possibilities exist in the random procedure. Either

- A value is randomly chosen according to the uniform distribution – i.e., all the values have the same chance
- A value is chosen in conformity with the value distribution – i.e., the most frequent value has the greatest chance of being selected

To illustrate the difference of the strategies Anyvalue and Random, consider this scheme. Let the attribute A have three possible values, V_1, V_2, V_3 with the relative distribution [0.5, 0.3, 0.2]. (Here, of course, we consider class-sensitive distribution for the learning phase, overall one for classification.)

Strategy Anyvalue for TDIDT replaces the missing value A =? by each possible value $A = V_j$, $j = 1, 2, 3$, and these selectors (attribute-value pairs) are utilized for selecting a new node (during learning), or pushed down along an existing decision tree (classification).

Strategy Anyvalue for covering algorithms: if the corresponding selector in a complex is for example, $A = V_3$ then the selector A =? in an instance is replaced by $A = V_3$, so that the matching always succeeds.

Let the pseudo-random number be for example, 0.4 in the strategy Random. Then, in the first case – i.e., uniform distribution (one can consider the relative distribution has been changed to [0.33, 0.33, 0.33]) – the missing value A =? is replaced by $A = V_2$. In the second possibility – i.e., the actual distribution – the missing value is replaced by $A = V_1$.

(M) Strategy Meta Fill In: Use Another Learning Topology for Substitution: This interesting strategy utilizes another ML algorithm in order to fill in the missing attribute values. This second (or *meta*) learning algorithm uses the remaining attribute values of a given example (instance, observation) for determining (inducing) the missing value of the attribute A_n. There are several approaches to this strategy.

The first one was designed by Breiman; it uses a *surrogate split* in order to determine the missing attribute value. We can observe that a surrogate attribute has the highest correlation with the original one.

Quinlan (1989) was the first to introduce the meta-fill-in strategy; in fact, this method was proposed by A. Shapiro during their private communication. It builds a decision tree for each attribute that attempts to derive a value of the attribute with a missing value for a given instance in terms of the values of other attributes of the given instance.

Lakshminarayan et al. (1996) introduced a more robust approach where a ML technique (namely, C4.5) is used to fill in the missing values.

Ragel and Cremilleux (1998) developed a fill-in strategy by using the association rules paradigm. It induces a set of association rules according to the entire training set. This method is able to efficiently process the missing attribute values.

Missing Value Processing Techniques in Various ML Paradigms

As mentioned above, various missing value processing techniques have been embedded into various ML paradigms. We introduce four such systems.

Quinlan (1986, 1989) applied missing value techniques into ID3, the most famous TDIDT (decision tree inducing) algorithm. His list exhibits two additional routines that were not discussed above:

- The evaluation of an attribute uses the routines *I*, *C*, *M*, and *R* (i.e., reduce the apparent information gain from evaluating an attribute by the proportion of training examples with the missing value for this attribute)
- When partitioning a training set using the selected attribute, the routines *I*, *U*, *C*, *F*, *A*, *M* were used
- The classification phase utilizes the strategies *U*, *C*, *F*, *M*, and *H* (i.e., halt the classification and assign the instance to the most likely class)

Quinlan then combined the above routine into triples each representing a different overall strategy; however, not all the possible combinations of these routines make sense.

His experiments revealed that the strategies starting with *R* or *C* behave reasonably accurately among them the strategy *RFF* is the best. Brazdil and Bruha (1992) improved this strategy for partitioning a training set. They combined the strategies *U* and *F*; therefore, they call it *R(UF)(UF)* strategy.

Bruha and Franek (1996) discusses the embedding of missing value strategies into the covering algorithm CN4 (Bruha and Kockova 1994), a large extension of the well-known CN2 (Clark and Niblett 1989). A condition of a decision rule has the form:

$$\text{Cmplx} = S_{q_1} \& \ldots \& S_{q_M}$$

where S_{qm}, $m = 1, \ldots, M$, is the mth selector testing the jth value V_j of the q_mth attribute, (i.e., exhibiting the form $x_{qm} = V_j$). For the purposes of processing missing values, we need to define the *matching ratio* of the example **x** and the rule's condition *Cond*. (Bruha and Franek 1996) uses two definitions:

- The product of matching ratios of its selectors:

$$\mu(x, \text{Cmplx}) = w(x) \prod_{m=1}^{M} \mu(x, S_{qm}) \qquad (2)$$

- or their average:

$$\mu(x, \text{Cmplx}) = \frac{w(x)}{M} \sum_{m=1}^{M} \mu(x, S_{qm}), \qquad (3)$$

where $w(\mathbf{x})$ is the weight of the example **x** (1 by default), and μ on the right-hand side is the selector's matching ratio (1).

The Naive Bayes algorithm can process missing attribute values in a very simple way, because the probabilities it works with are, in fact, the relative frequencies discussed above: the class-sensitive relative frequencies $f_{r,n,j}$ (for the learning phase) and the overall

relative frequencies $f_{n,j}$ (for the purposes of classification). When learning relative frequencies, all strategies can by applied. Only routine Fraction is useless because it copies the distribution of the rest of a training set. When classifying an instance with missing value A_n =?, all strategies can be applied as well. Section Fraction substitutes this instances with $J(n)$ instances by each known attribute value, and each "fractioned" instance is attached by the weight $f_{n,j}$, and classified separately.

Ragel and Cremilleux (1998) present the missing value processing strategy for the algorithm that induced ►association rules. Their algorithm uses a modified version of the routine *Ignore*. The instances with missing attribute values are not removed from the training database but the missing values are ignored (or "hidden").

The experiments with the above techniques for handling missing values have revealed the following. In both decision tree and decision rules inducing algorithms, the routine *Ignore* is evidently the worst strategy. An Interesting issue is that the association rule inducing algorithms use its modified version. In case of the decision tree inducing algorithms, the strategy *Fraction* is one of the best; however, the decision rules inducing algorithms found it not so efficient. The explanation for this fact is based on different ways of processing examples in these two paradigms: in TDIDT, all training examples are eventually incorporated into the decision tree generated by the learning algorithm; on the other hand, the covering paradigm algorithm generates rules that may not cover all of the examples from the training set (as some of the examples are found not to be representable).

Although the routine *Unknown* is one of the "winners" (at least in the rule inducing algorithms and Brazdil and Bruha (1992), it is not quite clear how one can interpret, on a philosophical as well as a semantic level, a branch in a decision tree or a decision rule that involves a selector with an attribute equal to "?" (missing value). Strategy Fraction can be faced by "problems": if an example /instance exhibits too many missing values, then this strategy generates too many "fractioned" examples with very negligible weights.

One can find out that each dataset has more or less its own "favorite" routine for processing missing attribute values. It evidently depends on the magnitude of noise and the source of unknownness in each dataset. The problem of a "favorite" strategy can be solved by various approaches. One possibility is to create a small "window" within a training set, and to check the efficiency of each strategy in this window, and then choose the most efficient one. Bruha (2003) discusses another possibility: investigating the advantages of utilizing the external background (domain-specific, expert) knowledge on an attribute hierarchical tree.

Also, the concept of the so-called ►meta-combiner (Fan, Chan & Stolfo, 1996) can be utilized. A learning algorithm processes a given training base for each strategy for missing values independently; thus, all the missing value strategies are utilized in parallel and the meta-classifier makes up its decision from the results of the base level (Bruha, 2004).

The above issue – i.e., selection or combination of various strategies for missing value processing – is an open field for future research.

Recommended Reading

Brazdil, P. B., & Bruha, I. (1992). A note on processing missing attribute values: A modified technique. *Workshop on Machine learning, Canadian Conference AI*, Vancouver.

Bruha, I. (2003). Unknown attribute value processing by domain-specific external expert knowledge. *7th WSEAS international conference on systems*, Corfu, Greek.

Bruha, I. (2004). Meta-learner for unknown attribute values processing: Dealing with inconsistency of meta-databases. *Journal of Intelligent Information Systems, 22*(1), 71–84.

Bruha, I., & Franek, F. (1996). Comparison of various routines for unknown attribute value processing: covering paradigm. *International Journal of Pattern Recognition and Artificial Intelligence, 10*(8), 939–955.

Bruha, I., & Berka, P. (2000). Discretization and fuzzification of numerical attributes in attribute-based learning. In P. S. Szczepaniak, P. J. G. Lisboa, & J. Kacprzyk (Eds.), *Fuzzy systems in medicine* (pp. 112–138). Physica, Springer.

Bruha, I., & Kockova, S. (1994). A support for decision making: Cost-sensitive learning system. *Artificial Intelligence in Medicine, 6*, 67–82.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3*, 261–283.

Fan, D. W., Chan, P. K., & Stolfo, S. J. (1996). A comparative evaluation of combiner and stacked generalization. *Workshop integrating multiple learning models, AAAI*, Portland.

Grzymala-Busse, J. W. (2003). Rough set strategies to date with missing attribute values, *Proceedings of workshop on foundations and new directions in data mining, IEEE Conference on data mining* (pp. 56–63).

Karmaker, A., & Kwer, S. (2005). Incorporating an EM-approach for handling missing attribute-values in decision tree induction. *International Conference on Hybrid Intelligent Systems*, 6–11.

Kononenko, I. (1992). Combining decisions of multiple rules. In B. du Boulay & V. Sgurev (Eds.), *Artificial intelligence V: Methodology, systems, applications* (pp. 87–96). Elsevier.

Lakshminarayan, K. et al. (1996). Imputation of missing data using machine learning techniques. *Conference Knowledge Discovery in Databases (KDD-96)*, 140–145.

Li, J., & Cercone, N. (2006). Assigning missing attribute values based on rough sets theory. *IEEE international conference on granular computing* (pp. 31–37). Atlanta.

Long, W. J., & Zhang, W. X. (2004). A novel measure of compatibility and methods of missing attribute values treatment in decision tables. *International Conference on Machine Learning and Cybernetics*, 2356–2360.

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

Quinlan, J. R. (1989). Unknown attribute values in ID3. *Proceedings of International Workshop on Machine Learning*, 164–168.

Ragel, A., & Cremilleux, B. (1998). Treatment of missing values for association rules. *Lecture Notes in Computer Science, 1394*, 258–270.

Missing Values

►Missing Attribute Values

Mistake-Bounded Learning

►Online Learning

Mixture Distribution

►Mixture Model

Mixture Model

Rohan A. Baxter
Australian Taxation Office

Synonyms

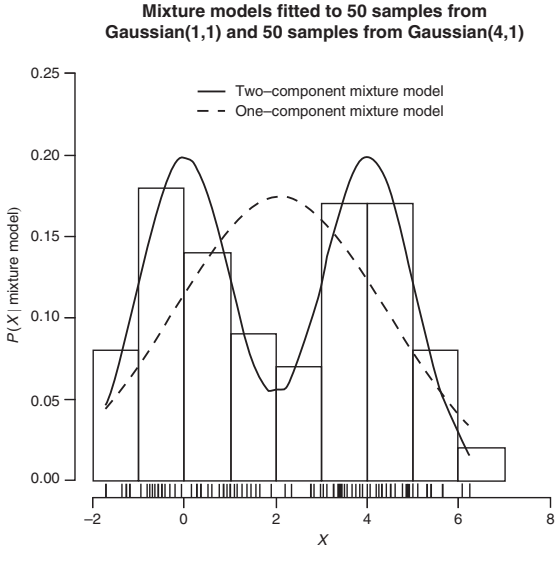
Finite mixture model; Latent class model; Mixture distribution; Mixture modeling

Definition

A mixture model is a collection of probability distributions or densities $D_1, \ldots, D_k$ and mixing weights or proportions $w_1, \ldots, w_k$, where k is the number of component distributions (Duda, Hart, & Stork, 2000; Lindsey, 1996; McLachlan & Peel, 2000).

The mixture model, $P(x \mid D_1, \ldots, D_k, w_1, \ldots, w_k) = \sum_{j=1}^{k} w_j P(x \mid D_j)$, is a probability distribution over the data conditional on the component distributions of the mixture and their mixing weights. It can be used for density estimation, model-based clustering or unsupervised learning, and classification.

Figure 1 shows one-dimensional data plotted along the x-axis with tick marks and a histogram of that data. The probability densities of two mixture models fitted to that data are then shown. The one-component mixture model is a Gaussian density with mean around 2 and standard deviation of 2.3. The two-component mixture model has one component with mean around 0 and the other with mean around 4, which reflects how these simple example data was artificially generated. This model can be used for clustering by considering each of its components as a cluster and assigning cluster membership based on the relative probability of a data item belonging to that component. Data less than 2 will have higher probability of belonging to the Gaussian with mean 0 component.



Mixture Model. Figure 1. Mixture model example for one-dimensional data

Motivation and Background

Mixture models are easy and convenient to apply. They trade off good power in data representation with relative ease in building the models. When used in clustering, a mixture model will have a component distribution covering each cluster, while the mixing weights reflect the relative proportion of a cluster's population. For example, a two-component mixture model of seal skull lengths from two different seal species may have one component with relative proportion 0.3 and the other with 0.7 reflecting the relative frequency of the two components.

Estimation

In order to use mixture models, the following choices need to be made by the modeler or by the mixture model software, based on the characteristics of a particular problem domain and its datasets:

1. The type of the component distributions (e.g., Gaussian, multinomial etc.)
2. The number of component distributions, k
3. The parameters for the component distributions (e.g., a one-dimensional Gaussian has a mean and standard deviation as parameters, a higher-dimensional Gaussian has a mean vector and covariance matrix as parameters)
4. Mixing weights, w_i
5. (Optional) component labels, c_j for each datum x_j, where $j = 1 \ldots n$ and n is the number of data

The fifth item above, component labels, are optional, because they are only used in latent class mixture model frameworks where a definite component membership is part of the model specification. Other mixture model frameworks use probabilistic membership of each datum to each component distribution and so do not need explicit component labels.

The most common way of fitting distribution parameters and mixture weights is to use the expectation-maximization (EM) algorithm to find the maximum likelihood estimates. The EM algorithm is an iterative algorithm that, starting with initial guesses of parameter values, computes the mixing weights (the expectation step). The next step is to then compute the parameter values based on these weights (the maximization step). The Expectation and Maximization steps iterate and convergence is assured (Redner & Walker, 2004). However, there is no guarantee that a global optimum has been found and so a number of random restarts may be required to find what other optima exist (Xu & Jordan, 1996).

As an alternative to random restarts, a good search strategy can be used to modify the current best solution, perhaps by choosing to split, merge, delete, or add, component distributions at random. This can also be a way to explore mixture models with different number of components (Figueiredo & Jain, 2002).

Since mixture models are a probabilistic model class, besides EM, other methods such as Bayesian methods or methods for graphical models can be used. These include Markov Chain Monte Carlo inference and Variational learning (Bishop, 2006).

Choosing the Number of Components

The number of components in a mixture model is often unknown when used for clustering real-world data. There have been many methods for choosing the number of components. The global maximum for maximum likelihood chooses a component for every data item, which is usually undesirable. Criteria based on information theory or Bayesian model selection choose reasonable numbers of components in many domains (McLachlan & Peel, 2000, Chap. 6, 5). There is no universally accepted method, because there is no universally accepted optimality criteria for clustering or density estimation. Use of an infinite mixture model, by using an infinite number of components, is one way to avoid the number of components problem (Rasmussen, 2000).

Types of Component Distributions

Besides Gaussian, other distributions can be used such as Poisson (for count data), von Mises (for data involving directions or angles), and Weibull. Heavy-tailed distributions require particular care, because standard estimation may not work when mean or variance is infinite (Dasgupta, Hopcroft, Kleinberg, & Sandler, 2005).

Another commonly needed variation is a mixture model to handle a mix of continuous and categorical features (McLachlan & Peel, 2000). For example, a binomial distribution can be used to model male/female

gender proportions and Gaussian to model length for data relating to a seal species sample.

A further extension is to allow components to depend on covariates, leading to mixtures of regression models (McLachlan & Peel, 2000). This leads to models such as mixtures of experts and hierarchical mixtures of experts (Bishop, 2006; McLachlan & Peel, 2000), which are flexible models for nonlinear regression. The combination of mixture models with Hidden Markov models allows the modeling of dependent data (McLachlan & Peel, 2000).

Large Datasets

The EM algorithm can be modified to find mixture models for very large datasets (Bradley, Reina, & Fayyad, 2000). The modification allows for a single scan of the data and involves identifying compressible regions of the data.

Theory

A key issue for mixture models is learnability (Chaudri, 2010). The more the component distributions overlap, the harder they are to learn. Higher-dimensional data also makes learning harder. Sometimes, these problems can be overcome by increasing the data quantity, but, in extremely hard cases, this will not work (Srebo, Shakhnarovich, & Roweis, 2006; Xu & Jordan, 1996).

Another issue is the relationship between adequate sample size and the number of components. A pragmatic policy is to set minimum mixing weights for component distributions. For example, for a dataset of size 100, if mixing weights are required to be greater than 0.1, this implies a maximum of ten components are possible to be learnt from the data with these parameter settings.

Applications

Mixture model software is often available in the clustering or density estimation parts of general statistical and data mining software. More specialized mixture modeling software for clustering data have included Autoclass (Autoclass, 2010), Snob (Snob, 2010), and mclust (Mclust, 2010).

Cross References

►Density-Based Clustering
►Density Estimation
►Gaussian Distribution
►Graphical Models
►Learning Graphical Models
►Markov Chain Monte Carlo
►Model-Based Clustering
►Unsupervised Learning

Recommended Reading

Autoclass, http://ti.arc.nasa.gov/project/autoclass/. Accessed 22 March 2010.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York: Springer.

Bradley, P. S., Reina, C. A., & Fayyad, U. M. (2000). Clustering very large databases using EM mixture models. Fifteenth International Conference on Pattern Recognition, (ICPR-2000), (Vol. 2., p. 2076), Barcelona, Spain.

Chaudri, K. (2009). Learning mixture models. http://themachinelearningforum.org/index.php/overviews/34-colt-overviews/53-learning-mixture-models.html. Accessed 21 March 2010.

Dasgupta, A., Hopcroft, J., Kleinberg, J., & Sandler, M. (2005). On learning mixtures of heavy-tailed distributions. In: Proceedings of the 46th Annual IEEE symposium on foundations of computer science, (FOCS '05), Pittsburgh, Pennsylvania, USA.

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern classification* (2nd ed.). New York: Wiley-Interscience.

Figueiredo, M. A. T., & Jain, A. T. (2002). Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*, 381–396.

Lindsey, B. G. (1996). *Mixture models: Theory, geometry and applications*. Hayward, CA: IMS.

McLachlan, G. J., & Peel, D. (2000). *Finite mixture models*. New York: Wiley.

Mclust, http://www.stat.washington.edu/mclust/. Accessed 22 March 2010.

Rasmussen, C. E. (2000). The infinite Gaussian mixture model, NIPS 12 (pp. 554–560). Cambridge, MA: MIT Press.

Redner, R. A., & Walker, H. F. (2004). Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review*, 26, 195–239.

Snob, http://www.datamining.monash.edu.au/software/snob/. Accessed 22 March 2010.

Srebo, N., Shakhnarovich, G., & Roweis, S. (2006). An investigation of computational and informational limits in Gaussian mixture modeling. In Proceedings of the 23rd international conference on machine learning (ICML 2006), Pittsburgh, Pennsylvania.

Xu, L., & Jordan, M. I. (1996). On convergence properties of the EM algorithm for Gaussian mixtures. *Neural Computation*, *8*, 129–151.

Mixture Modeling

►Mixture Model

Mode Analysis

►Density-Based Clustering

Model Evaluation

Geoffrey I. Webb
Monash University, Victoria, Australia

Model evaluation is the process of assessing a property or properties of a ►model.

Motivation and Background

It is often valuable to assess the efficacy of a model that has been learned. Such assessment is frequently relative—an evaluation of which of several alternative models is best suited to a specific application.

Processes and Techniques

There are many metrics by which a model may be assessed. The relative importance of each metric varies from application to application.

The primary considerations often relate to predictive efficacy—how useful will the predictions be in the particular context it is to be deployed. Measures relating to predictive efficacy include ►Accuracy, ►Lift, ►Mean Absolute Error, ►Mean Squared Error, ►Negative Predictive Value, ►Positive Predictive Value, ►Precision, ►Recall, ►Sensitivity, ►Specificity, and various metrics based on ►ROC analysis.

Computational issues may also be important, such as a model's size or its execution time.

In many applications one of the most important considerations is the ease with which the model can be understood by the users or how consistent it is with the users' prior beliefs and understanding of the application domain.

When assessing the predictive efficacy of a model learned from data, to obtain a reliable estimate of its likely performance on new data, it is essential that it is not assessed by considering its performance on the data from which it was learned. A learning algorithm must interpolate appropriate predictions for regions of the ►instance space that are not included in the training data. It is probable that the inferred model will be more accurate for those regions represented in the training data than for those that are not, and hence predictions are likely to be less accurate for instances that were not included in the training data. Estimates that have been computed on the training data are called ►*resubstitution estimates*. For example, the error of a model on the training data from which it was learned is called resubstitution error.

Algorithm evaluation techniques such as ►cross-validation, ►holdout evaluation, and ►bootstrap sampling are designed to provide more reliable estimates of the accuracy of the models learned by an algorithm than would be obtained by assessing them on the training data.

Cross References

►Algorithm Evaluation
►Overfitting
►ROC Analysis

Recommended Reading

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning*. New York: Springer.
Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.
Witten, I. H., Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed.). San Francisco: Morgan Kaufmann.

Model Selection

Model selection is the process of choosing an appropriate mathematical model from a class of models.

Model Space

►Hypothesis Space

Model Trees

Luís Torgo
University of Porto,
Porto, Portugal

Synonyms

Functional trees; Linear regression trees; Piecewise linear models

Definition

Model trees are supervised learning methods that obtain a type of tree-based ▶Regression model, similar to ▶Regression Trees, with the particularity of having functional models in the leaves instead of constants. These methods address multiple regression problems. In these problems we are usually given a training sample of n observations of a target continuous variable Y and of a vector of k predictor variables, $\mathbf{x} = X_1, \cdots, X_k$. Model trees provide an approximation of an unknown regression function $Y = f(\mathbf{x}) + \varepsilon$ with $Y \in R$ and ε, a normally distributed noise component with mean 0 and σ^2 variance. The leaves of these trees usually contain ▶linear regression models, although some works also consider other types of models.

Motivation and Background

Model trees are motivated by the purpose of overcoming some of the known limitations of regression trees caused by their piecewise constant approximation. In effect, by using constants at the leaves, regression trees provide a coarse grained function approximation leading to poor accuracy in some domains. Model trees try to overcome this by using more complex models on the leaves. Trees with linear models in the leaves were first considered in Breiman and Meisel (1976) and Friedman (1979). Torgo (1997) has extended the notion of model trees to other types of models in the tree leaves, namely, kernel regression, later extended to other types of local regression models (Torgo, 1999, 2000). The added complexity of the models used in the leaves increases the computational complexity of model trees when compared to regression trees, and also decreases their interpretability. In this context, several works Chaudhuri, Huang, Loh, & Yao (1994); Dobra & Gehrke (2002); Loh (2002); Malerba, Appice, Ceci, & Monopoli (2002); Natarajan & Pednault (2002); Torgo (2002); Malerba, Esposito, Ceci, & Appice (2004); Potts & Sammut (2005); Vogel, Asparouhov, & Scheffer (2007) have focused on obtaining model trees in a computationally efficient form.

Structure of Learning System

Approaches to model trees can be distinguished along two dimensions: the criterion used to select the best splits at each node, that is, the criterion guiding the partitioning obtained by the tree; and the type of models used in the leaves. The choices along the first dimension are mainly driven by considerations of computational efficiency. In effect, the selection of the best split node involves evaluating many candidate splits. The evaluation of a binary split (the most common splits in tree-based models) consists in calculating the error reduction produced by the split, that is,

$$\Delta(s,t) = Err(t) - \left(\frac{n_{t_L}}{n_t} \times Err(t_L) + \frac{n_{t_R}}{n_t} \times Err(t_R) \right) \quad (1)$$

where t is a tree node with sub-nodes t_L and t_R originated by the split test s, while n_t, n_{t_L}, n_{t_R} are the cardinalities of the respective sets of observations on each of these nodes, and $Err()$ is a function that estimates the error on a node being defined as,

$$Err(t) = \frac{1}{n_t} \sum_{\langle \mathbf{x}_i, y_i \rangle \in D_t} (y_i - g(D_t))^2 \quad (2)$$

where D_t is the sample of cases in node t, n_t is the cardinality of this set, and $g(D_t)$ is a function of the cases in node t.

In standard regression trees the function $g()$ is the average of the target variable Y, that is, $\frac{1}{n_t} \sum_{\langle \mathbf{x}_i, y_i \rangle \in D_t} y_i$. This corresponds to assuming a constant model on each leaf of the tree. The evaluation of each candidate split requires obtaining the models at the respective left and right branches (Eq. 1). If this model is an average, rather efficient incremental algorithms can be used to evaluate all candidate splits. On the contrary, if $g()$ is a ▶linear regression model or even other more complex models, this evaluation is not so simple and it is computationally very demanding, as a result of which systems that use this strategy (Karalic, 1992) become impractical for

large problems. In this context, several authors have adopted the alternative of growing the trees assuming constant values in the leaves and then fitting the complex models on each of the obtained leaves (e.g., Quinlan, 1992; Torgo, 1997, 1999, 2000). This only requires fitting as many models as there are leaves in the final tree. The main drawback of this approach is that the splits for the tree nodes are selected assuming the leaves will have averages instead of the models that in effect will be used. This may lead to splits that are suboptimal for the models that will fit on each leaf (Malerba et al., 2002, 2004). Several authors have tried to maintain the consistency of the split selection step with the models used in the leaves by proposing efficient algorithms for evaluating the different splits. In Malerba et al. (2002, 2004) linear models are obtained in a stepwise manner during tree growth. In Chaudhuri et al. (1994), Loh (2002), and Dobra and Gehrke (2002) the computational complexity is reduced by transforming the original regression problem into a classification problem. In effect, the best split is chosen by looking at the distribution of the sign of the residuals of a linear model fitted locally. In Natarajan and Pednault (2002); Torgo (2002); Vogel et al. (2007) the problem is addressed by proposing more efficient algorithms to evaluate all candidate splits. Finally, Potts and Sammut (2005) proposes an incremental algorithm to obtain model trees that fights the complexity of this task by imposing a limit on the number of splits that are considered for each node.

The most common form of model used in leaves is ►linear regression. Still, there are systems considering kernel models (Torgo, 1997), local linear models (Torgo, 1999), and partial linear models (Torgo, 2000). These alternatives provide smoother function approximation, although with increased computational costs and less interpretable models.

►Pruning in model trees does not bring any additional challenges when compared to standard regression trees and so similar methods are used for this over-fitting avoidance task. The same occurs with the use of model trees for obtaining predictions for new test cases. Each case is "dropped-down" the tree from the root node, following the branches according to the logical tests in the nodes, till a leaf is reached. The model in this leaf is used to obtain the prediction for the test case.

Cross References

►Random Forests
►Regression
►Regression Trees
►Supervised Learning
►Training Sample

Recommended Reading

Breiman, L., & Meisel, W. S. (1976). General estimates of the intrinsic variability of data in nonlinear regression models. *Journal of the American Statistical Association, 71*, 301–307.

Chaudhuri, P., Huang, M., Loh, W., & Yao, R. (1994). Piecewise-polynomial regression trees. *Statistica Sinica, 4*, 143–167.

Dobra, A., & Gehrke, J. E. (2002). Secret: A scalable linear regression tree algorithm. In *Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining*. Edmonton, Alberta, Canada: ACM.

Friedman, J. (1979). A tree-structured approach to nonparametric multiple regression. In T. Gasser & M. Rosenblatt (Eds.), *Smoothing techniques for curve estimation. Lecture notes in mathematics* (Vol. 757, pp. 5–22). Berlin/Heidelberg: Springer.

Karalic, A. (1992). Employing linear regression in regression tree leaves. In *Proceedings of ECAI-92*. New York, NY, USA: John Wiley & Sons, Inc.

Loh, W. (2002). Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica, 12*, 361–386.

Malerba, D., Appice, A., Ceci, M., & Monopoli, M. (2002). Trading-off local versus global effects of regression nodes in model trees. In *ISMIS '02: Proceedings of the 13th international symposium on foundations of intelligent systems* (pp. 393–402). Springer.

Malerba, D., Esposito, F., Ceci, M., & Appice, A. (2004). Top-down induction of model trees with regression and splitting nodes. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 26*(5), 612–625.

Natarajan, R., & Pednault, E. (2002). Segmented regression estimators for massive data sets. In *Proceedings of the second SIAM international conference on data mining (SDM'02)*. Arlington, VA, USA: SIAM.

Potts, D., & Sammut, C. (2005). Incremental learning of linear model trees. *Machine Learning, 61*(1–3), 5–48.

Quinlan, J. (1992). Learning with continuous classes. In Adams & Sterling (Ed.), *Proceedings of AI'92* (pp. 343–348). World Scientific.

Torgo, L. (1997). Functional models for regression tree leaves. In D. Fisher (Ed.), *Proceedings of the 14th international conference on machine learning*. San Francisco, CA, USA: Morgan Kaufmann.

Torgo, L. (1999). *Inductive learning of tree-based regression models*. PhD thesis, Faculty of Sciences, University of Porto.

Torgo, L. (2000). Partial linear trees. In P. Langley (Ed.), *Proceedings of the 17th international conference on machine learning (ICML 2000)* (pp. 1007–1014). San Francisco, CA, USA: Morgan Kaufmann.

Torgo, L. (2002). Computationally efficient linear regression trees. In K. Jajuga, A. Sokolowski, & H. Bock (Eds.), *Classification, clustering and data analysis: Recent advances and applications (Proceedings of IFCS 2002). Studies in classification, data analysis, and knowledge organization* (pp. 409–415). Heidelberg/New York: Springer.

Vogel, D., Asparouhov, O., & Scheffer, T. (2007). Scalable look-ahead linear regression trees. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 757–764). ACM.

Model-Based Clustering

Arindam Banerjee, Hanhuai Shan
University of Minnesota, Minneapolis, MN, USA

Definition

Model-based clustering is a statistical approach to data clustering. The observed (multivariate) data is assumed to have been generated from a finite mixture of component models. Each component model is a probability distribution, typically a parametric multivariate distribution. For example, in a multivariate Gaussian mixture model, each component is a multivariate Gaussian distribution. The component responsible for generating a particular observation determines the cluster to which the observation belongs. However, the component generating each observation as well as the parameters for each of the component distributions are unknown. The key learning task is to determine the component responsible for generating each observation, which in turn gives the clustering of the data. Ideally, observations generated from the same component are inferred to belong to the same cluster. In addition to inferring the component assignment of observations, most popular learning approaches also estimate the parameters of each component in the process. The strength and popularity of the methods stem from the fact that they are applicable for a wide variety of data types, such as multivariate, categorical, sequential, etc., as long as suitable component generative models can be constructed. Such methods have found applications in several domains such as text clustering, image processing, computational biology, and climate sciences.

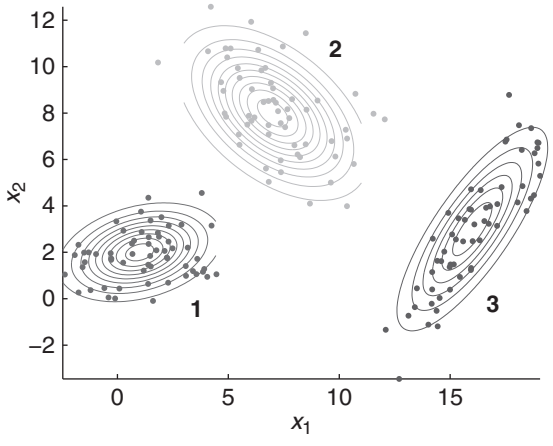
Structure of Learning System

Generative Model

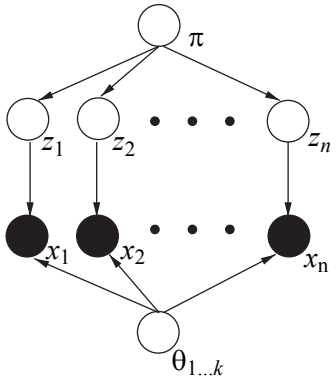
Let $X = \{x_1, \ldots, x_n\}$ be a dataset on which a k-clustering is to be performed. Let $p(x|\theta_1), \ldots, p(x|\theta_k)$ be k distributions which form the components of the mixture model from which the observed data is assumed to have been generated, and let $\pi = (\pi_1, \ldots, \pi_k)$ denote a prior distribution over the components. Then $\Theta = (\pi, \theta)$ constitutes the (unknown) parameters of the generative mixture model, where $\theta = \{\theta_1, \ldots, \theta_k\}$ and $\pi = \{\pi_1, \ldots, \pi_k\}$.

Given the model, an observation is assumed to be generated by the following two-step process: (1) randomly pick a component following the discrete distribution π over the components, i.e., the hth component is chosen with the probability of π_h; (2) the observation is sampled from the component distribution, e.g., if the hth component was chosen, we draw a sample $x \sim p(x|\theta_h)$. Each observation is assumed to be statistically independent so that they are all generated independently following the same two-step process.

Figure 1 gives an example of data drawn from a mixture of three ($k = 3$) 2-dimentional multivariate Gaussians. In the example, the discrete distribution over the component Gaussians is given by $\pi = (0.2, 0.3, 0.5)$. The parameter set $\theta_h, h = 1, 2, 3$ for any individual multivariate Gaussian consists of the mean vector μ_h and the covariance matrix Σ_h. For the example, we have $\mu_1 = [1, 2]$, $\mu_2 = [7, 8]$, $\mu_3 = [16, 3]$, and $\Sigma_1 = \begin{bmatrix} 3 & 0.5196 \\ 0.5196 & 1 \end{bmatrix}$, $\Sigma_2 = \begin{bmatrix} 4 & -1.7321 \\ -1.7321 & 3 \end{bmatrix}$, $\Sigma_3 = \begin{bmatrix} 3 & 3.0984 \\ 3.0984 & 5 \end{bmatrix}$.



Model-Based Clustering. Figure 1. Three 2-dimensional Gaussians



Model-Based Clustering. Figure 2. Bayesian network for a finite mixture model

The generative process could be represented as a Bayesian network as shown in Fig. 2, where the arrows denote the dependencies among variables/parameters. In the Bayesian network, (π, θ) are the parameters of the mixture model, x_i are the observations and z_i are the latent variables corresponding to the component which generates $x_i, i = 1, \ldots, n$. To generate an observation x_i, the model first samples a latent variable z_i from the discrete distribution π, and then samples the observation x_i from component distribution $p(x|\theta_{z_i})$.

Learning

Given a set of observations $X = \{x_1, \ldots, x_n\}$ assumed to have been generated from a finite mixture model, the learning task is to infer the latent variables z_i for each observation as well as estimate the model parameters $\Theta = (\pi, \theta)$. In the Gaussian mixture model example, the goal would be to infer the component responsible for generating each observation and estimate the mean and covariance for each component Gaussian as well as the discrete distribution π over the three Gaussians. After learning model parameters, the posterior probability $p(h|x_i, \Theta)$ of each observation x_i belonging to each component Gaussian gives a (soft) clustering for the observation.

The most popular approach for learning mixture models is based on maximum likelihood estimation (MLE) of the model parameters. In particular, given the set of observations X, one estimates the set of model parameters which maximizes the (log-)likelihood of observing the entire dataset X. For the finite mixture model, the likelihood of observing any data point x_i is given by

$$p(x_i|\Theta) = \sum_{h=1}^{k} \pi_h p(x_i|\theta_h) \,. \tag{1}$$

Since the data points in X are assumed to be statistically independent, the log-likelihood. (In practice, one typically focuses on maximizing the log-likelihood $\log p(X|\Theta)$ instead of the likelihood $p(X|\Theta)$ due to both numerical stability and analytical tractability). of observing the entire dataset X is given by

$$\begin{aligned} \log p(X|\Theta) &= \log \left(\prod_{i=1}^{n} p(x_i|\pi, \theta) \right) \\ &= \sum_{i=1}^{n} \log \left(\sum_{h=1}^{k} \pi_h p(x_i|\theta_h) \right) \,. \end{aligned} \tag{2}$$

A direct application of MLE is difficult since the log-likelihood cannot be directly optimized with respect to the model parameters. The standard approach to work around this issue is to use the expectation maximization (EM) algorithm which entails maximizing a tractable lower bound to the log-likelihood $\log p(X|\Theta)$. To this end, a latent variable z_i is explicitly introduced for each x_i to inform the component that x_i is generated from. The joint distribution of (x_i, z_i) is $p(x_i, z_i|\pi, \theta) = \pi_{z_i} p(x_i|\theta_{z_i})$. Let $Z = \{z_1, \ldots, z_n\}$ denote the set of latent variables corresponding to $X = \{x_1, \ldots, x_n\}$. The joint log-likelihood of (X, Z) then becomes

$$\begin{aligned} \log p(X, Z|\Theta) &= \sum_{i=1}^{n} \log p(x_i, z_i|\Theta) \\ &= \sum_{i=1}^{n} (\log \pi_{z_i} + \log p(x_i|\theta_{z_i})) \,. \end{aligned} \tag{3}$$

For a given set Z, it is easy to directly optimize (3) with respect to the parameters $\Theta = (\pi, \theta)$. However, Z is actually a random vector whose exact value is unknown. Hence, the log-likelihood $\log p(X, Z|\Theta)$ is a random variable depending on the distribution of Z. As a result, EM focuses on optimizing the following lower bound based on the expectation of $\log p(X, Z|\Theta)$ where the expectation is taken with respect to some distribution $p(Z)$ over the latent variable set Z. In particular, for any distribution $q(Z)$, we consider the lower bound

$$L(q, \Theta) = E_{Z \sim q}[\log p(X, Z|\theta)] + H(q(Z)) \,, \tag{4}$$

where the expectation on the first term is with respect to the posterior distribution $q(Z)$ and $H(q(Z))$ denotes the Shannon entropy of the latent variable set $Z \sim q(Z)$. A direct calculation shows that the difference between the true log-likelihood in (2) and the lower bound in (4) is exactly the relative entropy between $q(Z)$ and the posterior distribution $p(Z|X,\Theta)$, i.e.,

$$\log p(X|\Theta) - L(q,\Theta) = KL(q(Z)||p(Z|X,\Theta)) \geq 0 \quad (5)$$

$$\Rightarrow \log p(X|\Theta) \geq L(q,\Theta)\,, \quad (6)$$

where $KL(||)$ denotes the KL-divergence or relative entropy. As a result, when $q(Z) = p(Z|X,\Theta)$, the lower bound is exactly equal to the log-likelihood $\log p(X|\Theta)$. EM algorithms for learning mixture models work by alternately optimizing the lower bound $L(q,\Theta)$ over q and Θ. Starting with an initial guess $\Theta^{(0)}$ of the parameters, in iteration t such algorithms perform the following two steps:

E-step Maximize $L(q,\Theta^{(t-1)})$ with respect to $q(Z)$ to obtain

$$q^{(t)}(Z) = \underset{q(Z)}{argmax}\, L(q(Z),\Theta^{(t-1)}) = p(Z|X,\Theta^{(t-1)})\,. \quad (7)$$

M-step Maximize $L(q^{(t)},\Theta)$ with respect to Θ, i.e.,

$$\Theta^{(t)} = \underset{\Theta}{argmax}\, L(q^{(t)}(Z),\Theta)\,, \quad (8)$$

which is equivalent to

$$\Theta^{(t)} = \underset{\Theta}{argmax} \sum_{i=1}^{n} E_{z_i}[\log p(x_i,z_i|\Theta)]$$

since the second term in (4) does not depend on Θ.

Model-based clustering of multivariate data is often performed by learning a Mixture of Gaussians (MoG) using the EM algorithm. In a MoG model, the parameters corresponding to each component are the mean and covariance for each Gaussian given by (μ_h,Σ_h), $h = 1,\ldots,k$. For a given dataset X, the EM algorithm for learning MoG starts with an initial guess $\Theta^{(0)}$ for the parameters where $\Theta^{(0)} = \{(\pi_h^{(0)},\mu_h^{(0)},\Sigma_h^{(0)}), h = 1,\ldots,k\}$. At iteration t, the following updates are done:

E-step Update distributions over latent variables $z_i, i = 1,\ldots,n$ as

$$q^{(t)}(z_i = h) = p(z_i = h|x_i,\Theta^{(t-1)}) = \frac{\pi_h^{(t-1)} p(x_i|\mu_h^{(t-1)},\Sigma_h^{(t-1)})}{\sum_{h'=1}^{k} \pi_{h'}^{(t-1)} p(x_i|\mu_h^{(t-1)},\Sigma_{h'}^{(t-1)})}\,. \quad (9)$$

M-step Optimizing the lower bound over $\{(\pi_h,\mu_h,\Sigma_h), h = 1,\ldots,k\}$ yields

$$\pi_h^{(t)} = \frac{1}{n}\sum_{i=1}^{n} p(h|x_i,\Theta^{(t-1)})\,, \quad (10)$$

$$\mu_h^{(t)} = \frac{\sum_{i=1}^{n} x_i p(h|x_i,\Theta^{(t-1)})}{n\pi_h^{(t)}}\,, \quad (11)$$

$$\Sigma_h^{(t)} = \frac{\sum_{i=1}^{n}(x_i-\mu_h^{(t)})(x_i-\mu_h^{(t)})^T p(h|x_i,\Theta^{(t-1)})}{n\pi_h^{(t)}}\,. \quad (12)$$

The iterations are guaranteed to lead to monotonically non decreasing improvements of the lower bound $L(q,\Theta)$. The iterations are typically run till a suitable convergence criterion is satisfied. On convergence, one gets the estimates $\Theta = \{(\pi_h,\mu_h,\Sigma_h), h = 1,\ldots,k\}$ of the component parameters as well as the soft clustering $p(h|x_i,\Theta)$ of individual data points. The alternating maximization algorithm outlined above can get stuck in a local minima or saddle point of the objective function. In general, the iterations are not guaranteed to converge to a global optima. In fact, different initializations $\Theta^{(0)}$ of parameters can yield different final results. In practice, one typically tries a set of different initializations and picks the best among them according to the final value of the lower bound obtained. Extensive empirical research has gone into devising good initialization schemes for EM algorithm in the context of learning mixture models.

Recent years have seen progress in the design and analysis of provably correct algorithms for learning mixture models for certain well behaved distributions, where the component distributions are assumed to be

separated from each other in a well-defined sense. Such algorithms typically involve projecting data to a suitable lower-dimensional space where the components separate out and the clustering becomes simpler. One family of algorithms rely on random projections and are applicable to variety of problems including that of learning mixture of Gaussians. More recent developments include algorithms based on spectral projections and are applicable to any log-concave distributions.

Related Work

Model-based clustering is intimately related to a wide variety of centroid-based partitional clustering algorithms. In particular, the popular kmeans clustering algorithm can be viewed as a special case of learning mixture of Gaussians with a specific covariance structure. Given a dataset X, the kmeans problem is to find a partitioning $C = \{C_h, h = 1, \ldots, k\}$ of X such that the following objective is minimized:

$$J(C) = \sum_{h=1}^{k} \sum_{x \in C_h} \|x - \mu_h\|^2 ,$$

where μ_h is the mean of the points in C_h. Starting from an initial guess at the cluster means, the kmeans algorithm alternates between assigning points to the nearest cluster and updating the cluster means till convergence. Consider the problem of learning a mixture of Gaussians on X such that each Gaussian has a fixed covariance matrix $\Sigma_h = \beta I$, where I is the identity matrix and $\beta > 0$ is a constant. Then, as $\beta \to 0$, maximizing the scaled lower bound $\beta L(q, \Theta)$ corresponding to the mixture modeling problem becomes equivalent to minimizing the kmeans objective. Further, the EM algorithm outlined above reduces to the popular kmeans algorithm. In fact, such a reduction holds for a much larger class of centroid-based clustering algorithms based on Bregman divergences, which are a general class of divergence measures derived from convex function and have popular divergences such as squared Euclidean distance and KL-divergence as special cases. Centroid-based clustering with Bregman divergences can be viewed as a special case of learning mixtures of exponential family distributions with a reduction similar to the one from mixture of Gaussians to kmeans. Further, non linear clustering algorithms such as kernel kmeans can be viewed as a special case of learning mixture of Gaussians in a Hilbert space.

Recent years have seen generalizations of mixture models to mixed membership models and their non parametric extensions. Latent Dirichlet allocation is an example of such a mixed membership model for topic modeling in text corpora. The key novelty of mixed membership models is that they allow a different component proportions π_x for each observation x instead of a fixed proportion π as in mixture models. The added flexibility yields superior performance in certain problem domains.

Recommended Reading

Banerjee, A., Merugu, S., Dhillon, I., & Ghosh, J. (2005). Clustering with Bregman divergences. *Journal of Machine Learning Research, 6*, 1705–1749.

Bilmes, J. (1997). A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical Report ICSI-TR-97-02, University of Berkeley.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research, 3*, 993–1022.

Dasgupta, S. (1999). *Learning mixtures of Gaussians*. IEEE Symposium on foundations of Computer Science (FOCS). Washington, DC: IEEE Press.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological), 39*(1), 1–38.

Kannan, R., Salmasian, H., & Vempala, S. (2005). *The spectral method for general mixture models*. Conference on Learning Theory (COLT).

McLachlan, G. J., & Krishnan, T. (1996). *The EM algorithm and extensions*. NewYork: Wiley-Interscience.

McLachlan, G. J., & Peel, D. (2000). *Finite mixture models. Wiley series in probability and mathematical statistics: Applied probability and statistics section*. New York: Wiley.

Neal, R. M., & Hinton, G. E. (1998). A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan (Ed.), *Learning in graphical models* (pp. 355–368). Cambridge, MA: MIT Press.

Redner, R., & Walker, H. (1984). Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review, 26*(2), 195–239.

Model-Based Control

►Internal Model Control

Model-Based Reinforcement Learning

Soumya Ray[1], Prasad Tadepalli[2]
[1]Case Western Reserve University, Cleveland, OH, USA
[2]Oregon State University, Corvallis, OR, USA

Synonyms

Indirect reinforcement learning

Definition

Model-based Reinforcement Learning refers to learning optimal behavior indirectly by learning a model of the environment by taking actions and observing the outcomes that include the next state and the immediate reward. The models predict the outcomes of actions and are used in lieu of or in addition to interaction with the environment to learn optimal policies.

Motivation and Background

►Reinforcement Learning (RL) refers to learning to behave optimally in a stochastic environment by taking actions and receiving rewards (Sutton & Barto, 1998). The environment is assumed Markovian in that there is a fixed probability of the next state given the current state and the agent's action. The agent also receives an immediate reward based on the current state and the action. Models of the next-state distribution and the immediate rewards are referred to as "action models" and, in general, are not known to the learner. The agent's goal is to take actions, observe the outcomes including rewards and next states, and learn a policy or a mapping from states to actions that optimizes some performance measure. Typically the performance measure is the expected total reward in episodic domains, and the expected average reward per time step or expected discounted total reward in infinite-horizon domains.

The theory of ►Markov Decision Processes (MDPs) implies that under fairly general conditions, there is a stationary policy, i.e., a time-invariant mapping from states to actions, which maximizes each of the above reward measures. Moreover, there are MDP solution algorithms, e.g., value iteration and policy iteration (Puterman, 1994), which can be used to solve the MDP exactly given the action models. Assuming that the number of states is not exceedingly high, this suggests a straight-forward approach for model-based reinforcement learning. The models can be learned by interacting with the environment by taking actions, observing the resulting states and rewards, and estimating the parameters of the action models through maximum likelihood methods. Once the models are estimated to a desired accuracy, the MDP solution algorithms can be run to learn the optimal policy.

One weakness of the above approach is that it seems to suggest that a fairly accurate model needs to be learned over the entire domain to learn a good policy. Intuitively it seems that we should be able to get by without learning highly accurate models for suboptimal actions. A related problem is that the method does not suggest how best to explore the domain, i.e., which states to visit and which actions to execute to quickly learn an optimal policy. A third issue is one of scaling these methods, including model learning, to very large state spaces with billions of states.

The remaining sections outline some of the approaches explored in the literature to solve these problems.

Theory and Methods

Systems that solve MDPs using value-based methods can take advantage of models in at least two ways. First, with an accurate model, they can use offline learning algorithms that directly solve the modeled MDPs. Second, in an online setting, they can use the estimated models to guide exploration and action selection. Algorithms have been developed that exploit MDP models in each of these ways. We describe some such algorithms below.

Common approaches to solving MDPs given a model are value or policy iteration (Kaelbling, Littman, & Moore, 1996; Sutton & Barto, 1998). In these approaches, the algorithms start with a randomly initialized value function or policy. In value iteration, the algorithm loops through the state space, updating the value estimates of each state using Bellman backups, until convergence. In policy iteration, the algorithm calculates the value of the current policy and then loops through the state space, updating the current policy to be greedy with respect to the

backed up values. This is repeated until the policy converges.

When the model is unknown but being estimated as learning progresses, we could use value or policy iteration in the inner loop: after updating our current model estimate using an observed sample from the MDP, we could solve the updated MDP offline and take an action based on the solution. However, this is computationally very expensive. To gain efficiency, algorithms such as ▶Adaptive Real-time Dynamic Programming (ARTDP) (Barto, Bradtke, & Singh, 1995) and DYNA (Sutton, 1990) perform one or more Bellman updates using the action models after each real-world action and corresponding update to either a state-based or state-action-based value function. Other approaches, such as prioritized sweeping (Moore & Atkeson, 1993) and Queue-Dyna (Peng & Williams, 1993), have considered the problem of intelligently choosing which states to update after each iteration.

A different approach to discovering the optimal policy is to use algorithms that calculate the gradient of the utility measure with respect to some adjustable policy parameters. The standard policy gradient approaches that estimate the gradient from immediate rewards suffer from high variance due to the stochasticity of the domain and the policy. Wang and Dietterich propose a model-based policy gradient algorithm that alleviates this problem by learning a partial model of the domain (Wang & Dietterich, 2003). The partial model is solved to yield the value function of the current policy and the expected number of visits to each state, which are then used to derive the gradient of the policy in closed form. The authors observe that their approach converges in many fewer exploratory steps compared with model-free policy gradient algorithms in a number of domains including a real-world resource-controlled scheduling problem.

One of the many challenges in model-based reinforcement learning is that of efficient exploration of the MDP to learn the dynamics and the rewards. In the "Explicit Explore and Exploit" or E^3 algorithm, the agent explicitly decides between exploiting the known part of the MDP and optimally trying to reach the unknown part of the MDP (exploration) (Kearns & Singh, 2002). During exploration, it uses the idea of "balanced wandering," where the least executed action in the current state is preferred until all actions are executed a certain number of times. In contrast, the R-Max algorithm implicitly chooses between exploration and exploitation by using the principle of "optimism under uncertainty" (Brafman & Tennenholtz, 2002). The idea here is to initialize the model parameters optimistically so that all unexplored actions in all states are assumed to reach a fictitious state that yields maximum possible reward from then on regardless of which action is taken. Both these algorithms are guaranteed to find models whose approximate policies are close to the optimal with high probability in time polynomial in the size and mixing time of the MDP.

Since a table-based representation of the model is impractical in large state spaces, efficient model-based learning depends on compact parameterization of the models. Dynamic Bayesian networks offer an elegant way to represent action models compactly by exploiting conditional independence relationships, and have been shown to lead to fast convergence of models (Tadepalli & Ok, 1998). In some cases, choosing an appropriate prior distribution over model parameters can be important and lead to faster learning. In recent work, the acquisition of a model prior has been investigated in a multi-task setting (Wilson, Fern, Ray, & Tadepalli, 2007). In this work, the authors use a hierarchical Bayesian model to represent classes of MDPs. Given observations from a new MDP, the algorithm uses the model to infer an appropriate class (creating a new class if none seem appropriate). It then uses the distributions governing the inferred class as a prior to guide exploration in the new MDP. This approach is able to significantly speed up the rate of convergence to optimal policy as more environments are seen.

In recent work, researchers have explored the possibility of using approximate models coupled with policy gradient approaches to solve hard control problems (Abbeel, Quigley, & Ag, 2006). In this work, the approximate model is used to calculate gradient directions for the policy parameters. When searching for an improved policy, however, the real environment is used to calculate the utility of each intermediate policy. Observations from the environment are also used to update the approximate model. The authors show that their approach improves upon model-based algorithms which only used the approximate model while learning.

Applications

In this section, we describe some domains where model-based reinforcement learning has been applied.

Model-based approaches have been commonly used in RL systems that play two-player games (Baxter, Tridgell, & Weaver, 1998; Tesauro, 1995). In such systems, the model corresponds to legal moves in the game. Such models are easy to acquire and can be used to perform lookahead search on the game tree. For example, the TD-LEAF(λ) system (Baxter et al., 1998) uses the values at the leaves of an expanded game tree at some depth to update the estimate of the value of the current state. After playing a few hundred chess games, this algorithm was able to reach the play level of a US Master.

Model-based reinforcement learning has been used in a spoken dialog system (Singh, Kearns, Litman, & Walker, 1999). In this application, a dialog is modeled as a turn-based process, where at each step the system speaks a phrase and records certain observations about the response and possibly receives a reward. The system estimates a model from the observations and rewards and uses value iteration to compute optimal policies for the estimated MDP. The authors show empirically that, among other things, the system finds sensible policies and is able to model situations that involve "distress features" that indicate the dialog is in trouble.

It was shown that in complex real-world control tasks such as pendulum swing-up task on a real anthropomorphic robot arm, model-based learning is very effective in learning from demonstrations (Atkeson & Schaal, 1997). A model is learned from the human demonstration of pendulum swing-up, and an optimal policy is computed using a standard approach in control theory called linear quadratic regulation. Direct imitation of the human policy would not work in this case due to the small differences in the tasks and the imperfections of the robot controller. On the other hand, model-based learning was able to learn successfully from short demonstrations of pendulum swing up. However, on a more difficult swing-up task that includes pumping, model-based learning by itself was inadequate due to the inaccuracies in the model. They obtained better results by combining model-based learning with learning appropriate task parameters such as the desired pendulum target angle at an intermediate stage where the pendulum was at its highest point.

In more recent work, model-based RL has been used to learn to fly a remote-controlled helicopter (Abbeel, Coates, Quigley, & Ng, 2007). Again, the use of model-free approaches is very difficult, because almost any random exploratory action results in an undesirable outcome (i.e., a crash). To learn a model, the system bootstraps from a trajectory that is observed by watching an expert human fly the desired maneuvers. In each step, the system learns a model with the observed trajectory and finds a controller that works in simulation with the model. This controller is then tried with the real helicopter. If it fails to work well, the model is refined with the new observations and the process is repeated. Using this approach, the system is able to learn a controller that can repeatedly perform complex aerobatic maneuvers, such as flips and rolls.

Model-based RL has also been applied to other domains, such as robot juggling (Schaal & Atkeson, 1994) and job-shop scheduling (Zhang & Dietterich, 1995). Some work has also been done that compares model-free and model-based RL methods (Atkeson & Santamaria, 1997). From their experiments, the authors conclude that, for systems with reasonably simple dynamics, model-based RL is more data efficient, finds better policies, and handles changing goals better than model-free methods. On the other hand, model-based methods are subject to errors due to inaccurate model representations.

Future Directions

Representing and learning richer action models for stochastic domains that involve relations, numeric quantities, and parallel, hierarchical, and durative actions is a challenging open problem. Efficient derivation of optimal policies from such rich representations of action models is another problem that is partially explored in ►symbolic dynamic programming. Constructing good policy languages appropriate for a given action model or class of models might be useful to accelerate learning near-optimal policies for MDPs.

Cross References

►Adaptive Real-Time Dynamic Programming
►Autonomous Helicopter Flight Using Reinforcement Learning
►Bayesian Reinforcement Learning

►Efficient Exploration in Reinforcement Learning
►Symbolic Dynamic Programming

Recommended Reading

Abbeel, P., Coates, A., Quigley, M., & Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems* (Vol. 19, pp. 1–8). Cambridge, MA: MIT Press.

Abbeel, P., Quigley, M., & Ng, A. Y. (2006). Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on machine learning* (pp. 1–8). ACM Press, New York, USA.

Atkeson, C. G., & Santamaria, J. C. (1997). A comparison of direct and model-based reinforcement learning. In *Proceedings of the international conference on robotics and automation* (pp. 20–25). IEEE Press.

Atkeson, C. G., & Schaal, S. (1997). Robot learning from demonstration. In *Proceedings of the fourteenth international conference on machine learning* (Vol. 4, pp. 12–20). San Francisco: Morgan Kaufmann.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, 72*(1), 81–138.

Baxter, J., Tridgell, A., & Weaver, L. (1998). TDLeaf(λ): Combining temporal difference learning with game-tree search. In *Proceedings of the ninth Australian conference on neural networks (ACNN'98)* (pp. 168–172).

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research, 2*, 213–231.

Kaelbling, L. P., Littman, M. L., & Moore, A. P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Kearns, M., & Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning, 49*(2/3), 209–232.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning, 13*, 103–130.

Peng, J., & Williams, R. J. (1993). Efficient learning and planning within the dyna framework. *Adaptive Behavior, 1*(4), 437–454.

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

Schaal, S., & Atkeson, C. G. (1994). Robot juggling: Implementation of memory-based learning. *IEEE Control Systems Magazine, 14*(1), 57–71.

Singh, S., Kearns, M., Litman, D., & Walker, M. (1999) Reinforcement learning for spoken dialogue systems. In *Advances in neural information processing systems* (Vol. 11, pp. 956–962). MIT Press.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning* (pp. 216–224). San Francisco: Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Tadepalli, P., & Ok, D. (1998). Model-based average-reward reinforcement learning. *Artificial Intelligence, 100*, 177–224.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM, 38*(3), 58–68.

Wang, X., & Dietterich, T. G. (2003). Model-based policy gradient reinforcement learning. In *Proceedings of the 20th international conference on machine learning* (pp. 776–783). AAAI Press.

Wilson, A., Fern, A., Ray, S., & Tadepalli, P. (2007). Multi-task reinforcement learning: A hierarchical Bayesian approach. In *Proceedings of the 24th international conference on machine learning* (pp. 1015–1022). Madison, WI: Omnipress.

Zhang, W., & Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In *Proceedings of the international joint conference on artificial intelligence* (pp. 1114–1120). Morgan Kaufman.

Modularity Detection

►Group Detection

MOO

►Multi-Objective Optimization

Morphosyntactic Disambiguation

►POS Tagging

Most General Hypothesis

Synonyms

Maximally general hypothesis

Definition

A hypothesis, h, is a most general hypothesis if it covers none of the negative examples and there is no other hypothesis h' that covers no negative examples, such that h is strictly more specific than h'.

Cross References

►Learning as Search

Most Similar Point

►Nearest Neighbor

Most Specific Hypothesis

Synonyms
Maximally specific hypothesis

Definition
A hypothesis, h, is a most specific hypothesis if it covers none of the negative examples and there is no other hypothesis h' that covers no negative examples, such that h is strictly more general than h'.

Cross References
►Learning as Search

Multi-Agent Learning I: Problem Definition

Yoav Shoham, Rob Powers
Stanford University,
Stanford, CA, USA

Definition
Multi-agent learning (MAL) refers to settings in which multiple agents learn simultaneously. Usually defined in a game theoretic setting, specifically in repeated games or stochastic games, the key feature that distinguishes multi-agent learning from single-agent learning is that in the former the learning of one agent impacts the learning of others. As a result, neither the problem definition for multi-agent learning nor the algorithms offered follow in a straightforward way from the single-agent case. In this first of two entries on the subject we focus on the problem definition.

Background
The topic of multi-agent learning (MAL henceforth) has a long history in game theory, almost as long as the history of game theory itself (Another more recent term for the area within game theory is *interactive learning*). In artificial intelligence (AI) the history of *single*-agent learning is of course as rich if not richer; one need not look further than this Encyclopedia for evidence. And while it is only in recent years that AI has branched into the multi-agent aspects of learning, it has done so with something of a vengeance. If in 2003 one could describe the AI literature on MAL by enumerating the relevant articles, today this is no longer possible. The leading conferences routinely feature articles on MAL, as do the journals (We acknowledge a simplification of history here. There is definitely MAL work in AI that predates the last few years, though the relative deluge is indeed recent. Similarly, we focus on AI since this is where most of the action is these days, but there are also other areas in computer science that feature MAL material; we mean to include that literature here as well).

While the AI literature maintains a certain flavor that distinguishes it from the game theoretic literature, the commonalities are greater than the differences. Indeed, alongside the area of mechanism design, and perhaps the computational questions surrounding solution concepts such as the Nash equilibrium, MAL is today arguably one of the most fertile interaction grounds between computer science and game theory. The key aspect of MAL, which ties the work together, and which distinguishes it from single-agent learning, is the fact that in MAL one cannot separate the process of learning from the process of teaching. The learning of one agent causes it to change its behavior; this causes other agents to adapt their behavior, which in turn causes the first agent to keep adapting too. Such reciprocal – or interactive – learning calls not only for different types of learning algorithms, but also for different yardsticks by which to evaluate learning. For this reason, the literature on MAL can be confusing. Not only do the learning techniques vary, but the goal of learning and the evaluation measures are diverse, and often left only implicit.

We will couch our discussion in the formal setting of *stochastic games* (a.k.a. *Markov games*). Most of the MAL literature adopts this setting, and indeed most of it focuses on the even more narrow class of *repeated games*. Furthermore, stochastic games also generalize *Markov decision problems* (MDPs), the setting from

which much of the relevant learning literature in AI originates. These are defined as follows.

A stochastic game can be represented as a tuple: $(N, S, \vec{A}, \vec{R}, T)$. N is a set of agents indexed $1, \ldots, n$. S is a set of n-agent stage games. $\vec{A} = A_1, \ldots, A_n$, with A_i the set of actions (or pure strategies) of agent i (note that we assume the agent has the same strategy space in all games; this is a notational convenience, but not a substantive restriction). $\vec{R} = R_1, \ldots, R_n$, with $R_i : S \times \vec{A} \to \mathcal{R}$ giving the immediate reward function of agent i for stage game S. $T : S \times \vec{A} \to \Pi(S)$ is a stochastic transition function, specifying the probability of the next stage game to be played based on the game just played and the actions taken in it.

We also need to define a way for each agent to aggregate the set of immediate rewards received in each state. For finitely repeated games we can simply use the sum or average, while for infinite games the most common approaches are to use either the limit average or the sum of discounted awards $\sum_{t=1}^{\infty} \delta^t r_t$, where r_t is the reward received at time t.

A repeated game is a stochastic game with only one stage game, while an MDP is a stochastic game with only one agent. (Note: While most of the MAL literature lives happily in this setting, we would be remiss not to acknowledge the literature that does not. Certainly, one could discuss learning in the context of extensive-form games of incomplete and/or imperfect information. Even farther afield, interesting studies of learning exist in large population games and evolutionary models, particularly *replicator dynamics (RD)* and *evolutionary stable strategies (ESS)*.)

What is there to learn in stochastic games? Here we need to be explicit about some aspects of stochastic games that were glossed over so far. Do the agents know the stochastic game, including the stage games and the transition probabilities? If not, do they at least know the specific game being played at each stage, or only the actions available to them? What do they see after each stage game has been played – only their own rewards, or also the actions played by the other agent(s)? Do they perhaps magically see the other agent(s)' mixed strategy in the stage game? And so on.

In general, games may be known or not, play may be observable or not, and so on. We will focus on known, fully observable games, where the other agent's strategy (or agents' strategies) is not known a priori (though in some cases there is a prior distribution over it). In our restricted setting there are two possible things to learn. First, the agent can learn the opponent's (or opponents') strategy (or strategies), so that the agent can then devise a best (or at least a good) response. Alternatively, the agent can learn a strategy of his own that does well against the opponents, without explicitly learning the opponent's strategy. The first is sometimes called *model-based learning*, and the second *model-free learning*.

In broader settings there is more to learn. In particular, with unknown games, one can learn the game itself. Some will argue that the restricted setting is not a true learning setting, but (a) much of the current work on MAL, particularly in game theory, takes place in this setting, and (b) the foundational issues we wish to tackle surface already here. In particular, our comments are intended to also apply to the work in the AI literature on games with unknown payoffs, work which builds on the success of learning in unknown MDPs. We will have more to say about the nature of "learning" in the setting of stochastic games in the following sections.

Problem Definition

When one examines the MAL literature one can identify several distinct agendas at play, which are often left implicit and conflated. A prerequisite for success in the field is to be very explicit about the problem being addressed. Here we list five distinct coherent goals of MAL research. They each have a clear motivation and a success criterion. They can be caricatured as follows:

1. Computational
2. Descriptive
3. Normative
4. Prescriptive, cooperative
5. Prescriptive, non-cooperative

The first agenda is computational in nature. It views learning algorithms as an iterative way to compute properties of the game, such as solution concepts. As an example, fictitious play was originally proposed as a way of computing a sample Nash equilibrium for zero-sum games, and replicator dynamics has been proposed

for computing a sample Nash equilibrium in symmetric games. These tend not to be the most efficient computation methods, but they do sometimes constitute quick-and-dirty methods that can easily be understood and implemented.

The second agenda is descriptive – it asks how natural agents learn in the context of other learners. The goal here is to investigate formal models of learning that agree with people's behavior (typically, in laboratory experiments), or possibly with the behaviors of other agents (e.g., animals or organizations). This problem is clearly an important one, and when taken seriously calls for strong justification of the learning dynamics being studied. One approach is to apply the experimental methodology of the social sciences.

The centrality of equilibria in game theory underlies the third agenda we identify in MAL, which for lack of a better term we called normative, and which focuses on determining which sets of learning rules are in equilibrium with each other. More precisely, we ask which repeated-game strategies are in equilibrium; it just so happens that in repeated games, most strategies embody a learning rule of some sort. For example, we can ask whether fictitious play and Q-learning, appropriately initialized, are in equilibrium with each other in a repeated Prisoner's Dilemma game.

The last two agendas are prescriptive; they ask how agents *should* learn. The first of these involves distributed control in dynamic systems. There is sometimes a need or desire to decentralize the control of a system operating in a dynamic environment, and in this case the local controllers must adapt to each other's choices. This direction, which is most naturally modeled as a repeated or stochastic common-payoff (or "team") game. Proposed approaches can be evaluated based on the value achieved by the joint policy and the resources required, whether in terms of computation, communication, or time required to learn the policy. In this case there is rarely a role for equilibrium analysis; the agents have no freedom to deviate from the prescribed algorithm.

In our final agenda, termed "prescriptive, noncooperative," we ask how an agent should act to obtain high reward in the repeated (and more generally, stochastic) game. It thus retains the design stance of AI, asking how to design an optimal (or at least effective) agent for a given environment. It just so happens that this environment is characterized by the types of agents inhabiting it, agents who may do some learning of their own. The objective of this agenda is to identify effective strategies for environments of interest. An effective strategy is one that achieves a high reward in its environment, where one of the main characteristics of this environment is the selected class of possible opponents. This class of opponents should itself be motivated as being reasonable and containing opponents of interest. Convergence to an equilibrium is not a goal in and of itself.

Recommended Reading

Requisite background in game theory can be obtained from the many introductory texts, and most compactly from Leyton-Brown and Shoham (2008). Game theoretic work on multi-agent learning is covered in Fudenberg and Levine (1998) and Young (2004). An expanded discussion of the problems addressed under the header of MAL can be found in Shoham et al. (2007), and the responses to it in Vohra and Wellman (2007). Discussion of MAL algorithms, both traditional and more novel ones, can be found in the above references, as well as in Greenwald and Littman (2007).

Fudenberg, D., & Levine, D. (1998). *The theory of learning in games*. Cambridge: MIT Press.

Greenwald, A., & Littman, M. L. (Eds.). (2007). Special issue on learning and computational game theory. *Machine Learning 67*(1–2).

Leyton-Brown, K., & Shoham, Y. (2008). *Essentials of game theory*. San Rafael, CA: Morgan and Claypool.

Shoham, Y., Powers, W. R., & Grenager, T. (2007). If multiagent learning is the answer, what is the question? *Artificial Intelligence, 171*(1), 365–377. Special issue on foundations of multiagent learning.

Vohra, R., & Wellman, M. P. (Eds.). (2007). Special issue on foundations of multiagent learning. *Artificial Intelligence, 171*(1).

Young, H. P. (2004). *Strategic learning and its limits*. Oxford: Oxford University Press.

Multi-Agent Learning II: Algorithms

Yoav Shoham, Rob Powers
Stanford University, Stanford, CA, USA

Definition

Multi-agent learning (MAL) refers to settings in which multiple agents learn simultaneously. Usually defined in a game theoretic setting, specifically in repeated games or stochastic games, the key feature that distinguishes

MAL from single-agent learning is that in the former the learning of one agent impacts the learning of others. As a result, neither the problem definition for multi-agent learning nor the algorithms offered follow in a straightforward way from the single-agent case. In this second of two entries on the subject we focus on algorithms.

Some MAL Techniques

We will discuss three classes of techniques – one representative of work in game theory, one more typical of work in artificial intelligence (AI), and one that seems to have drawn equal attention from both communities.

Model-Based Approaches

The first approach to learning we discuss, which is common in the game theory literature, is the model-based one. It adopts the following general scheme:

1. Start with some model of the opponent's strategy.
2. Compute and play the best response.
3. Observe the opponent's play and update your model of his/her strategy.
4. Go to step 2.

Among the earliest, and probably the best-known, instance of this scheme is *fictitious play*. The model is simply a count of the plays by the opponent in the past. The opponent is assumed to be playing a stationary strategy, and the observed frequencies are taken to represent the opponent's mixed strategy. Thus after five repetitions of the Rochambeau game (R) in which the opponent played (R, S, P, R, P), the current model of his/her mixed strategy is $R = 0.4, P = 0.4, S = 0.2$.

There exist many variants of the general scheme, for example, those in which one does not play the exact best response in step 2. This is typically accomplished by assigning a probability of playing each pure strategy, assigning the best response the highest probability, but allowing some chance of playing any of the strategies. A number of proposals have been made of different ways to assign these probabilities such as *smooth fictitious play* and *exponential fictitious play*.

A more sophisticated version of the same scheme is seen in *rational learning*. The model is a distribution over the repeated-game strategies. One starts with some prior distribution; for example, in a repeated Rochambeau game, the prior could state that with probability 0.5 the opponent repeatedly plays the equilibrium strategy of the stage game, and, for all $k > 1$, with probability 2^{-k} she plays R k times and then reverts to the repeated equilibrium strategy. After each play, the model is updated to be the posterior obtained by Bayesian conditioning of the previous model. For instance, in our example, after the first non-R play of the opponent, the posterior places probability 1 on the repeated equilibrium play.

Model-Free Approaches

An entirely different approach that has been commonly pursued in the AI literature is the model-free one, which avoids building an explicit model of the opponent's strategy. Instead, over time one learns how well one's own various possible actions fare. This work takes place under the general heading of *reinforcement learning* (we note that the term is used somewhat differently in the game theory literature), and most approaches have their roots in the Bellman equations. We start our discussion with the familiar single-agent *Q-learning* algorithm for computing an optimal policy in an unknown Markov Decision Problem (MDP).

$$Q(s,a) \leftarrow (1-\alpha_t)Q(s,a) + \alpha_t[R(s,a) + \gamma V(s')]$$

$$V(s) \leftarrow \max_{a \in A} Q(s,a).$$

As is well known, with certain assumptions about the way in which actions are selected at each state over time and constraints on the learning rate schedule, α_t, Q-learning can be shown to converge to the optimal value function V^*.

The Q-learning algorithm can be extended to the multi-agent stochastic game setting by having each agent simply ignore the other agents and pretend that the environment is passive:

$$Q_i(s,a_i) \leftarrow (1-\alpha_t)Q_i(s,a_i) + \alpha_t[R_i(s,\vec{a}) + \gamma V_i(s')]$$

$$V_i(s) \leftarrow \max_{a_i \in A_i} Q_i(s,a_i).$$

Several authors have tested variations of the basic Q-learning algorithm for MAL. However, this approach ignores the multi-agent nature of the setting entirely. The Q-values are updated without regard for the actions selected by the other agents. While this can be justified when the opponents' distributions of actions are

stationary, it can fail when an opponent may adapt its choice of actions based on the past history of the game.

A first step in addressing this problem is to define the Q-values as a function of all the agents' actions:

$$Q_i(s,\vec{a}) \leftarrow (1-\alpha)Q_i(s,\vec{a}) + \alpha[R_i(s,\vec{a}) + \gamma V_i(s')].$$

We are, however, left with the question of how to update V, given the more complex nature of the Q-values.

For (by definition, two-player) zero-sum Stochastic Games (SGs), the *minimax-Q* learning algorithm updates V with the minimax of the Q-values:

$$V_1(s) \leftarrow \max_{P_1 \in \Pi(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} P_1(a_1) Q_1(s,(a_1,a_2)).$$

Later work proposed other update rules for the Q and V functions focusing on the special case of common-payoff (or "team") games. A stage game is common-payoff if at each outcome all agents receive the same payoff. The payoff is, in general, different in different outcomes, and thus the agents' problem is that of coordination; indeed, these are also called *games of pure coordination*.

The work on zero-sum and common-payoff games continues to be refined and extended; much of this work has concentrated on provably optimal tradeoffs between exploration and exploitation in unknown, zero-sum games. Other work attempted to extend the "Bellman heritage" to general-sum games (as opposed to zero-sum or common-payoff games), but the results here have been less conclusive.

Regret Minimization Approaches

Our third and final example of prior work in MAL is no-regret learning. It is an interesting example for two reasons. First, it has some unique properties that distinguish it from the work above. Second, both the AI and game theory communities appear to have converged on it independently. The basic idea goes back to early work on how to evaluate the success of learning rules in the mid-1950s, and has since been extended and rediscovered numerous times over the years under the names of universal consistency, no-regret learning, and the Bayes' envelope. The following algorithm is a representative of this body of work. We start by defining the *regret*, $r_i^t(a_j, s_i)$ of agent i for playing the sequence of actions s_i instead of playing action a_j, given that the opponents played the sequence s_{-i}.

$$r_i^t(a_j, s_i | s_{-i}) = \sum_{k=1}^{t} R\left(a_j, s_{-i}^k\right) - R\left(s_i^k, s_{-i}^k\right).$$

The agent then selects each of its actions with probability proportional to $\max\left(r_i^t(a_j, s_i), 0\right)$ at each time step $t+1$.

Some Typical Results

One sees at least three kinds of results in the literature regarding the learning algorithms presented above, and others similar to them. These are:

1. Convergence of the strategy profile to an (e.g., Nash) equilibrium of the stage game in self-play (i.e., when all agents adopt the learning procedure under consideration).
2. Successful learning of an opponent's strategy (or opponents' strategies).
3. Obtaining payoffs that exceed a specified threshold.

Each of these types comes in many flavors; here are some examples. The first type is perhaps the most common in the literature, in both game theory and AI. For example, while fictitious play does not in general converge to a Nash equilibrium of the stage game, the distribution of its play can be shown to converge to an equilibrium in zero-sum games, 2×2 games with generic payoffs, or games that can be solved by iterated elimination of strictly dominated strategies. Similarly in AI, minimax-Q learning is proven to converge in the limit to the correct Q-values for any zero-sum game, guaranteeing convergence to a Nash equilibrium in self-play. This result makes the standard assumptions of infinite exploration and the conditions on learning rates used in proofs of convergence for single-agent Q-learning.

Rational learning exemplifies results of the second type. The convergence shown is to correct beliefs about the opponent's repeated game strategy; thus it follows that, since each agent adopts a best response to their beliefs about the other agent, in the limit the agents will converge to a Nash equilibrium of the repeated game. This is an impressive result, but it is limited by two factors: the convergence depends on a very strong

assumption of absolute continuity; and the beliefs converged to are correct only with respect to the aspects of history that are observable given the strategies of the agents. This is an involved topic, and the reader is referred to the literature for more details.

The literature on no-regret learning provides an example of the third type of result, and has perhaps been the most explicit about criteria for evaluating learning rules. For example, one pair of criteria that have been suggested are as follows. The first criterion is that the learning rule should be "safe," which is defined as the requirement that the learning rule must guarantee at least the minimax payoff of the game. (The minimax payoff is the maximum expected value a player can guarantee against any possible opponent.) The second criterion is that the rule should be "consistent." In order to be "consistent," the learning rule must guarantee that it does at least as well as the best response to the empirical distribution of play when playing against an opponent whose play is governed by independent draws from a fixed distribution. "Universal consistency" is then defined as the requirement that a learning rule does at least as well as the best response to the empirical distribution regardless of the actual strategy the opponent is employing (this implies both safety and consistency). The requirement of "universal consistency" is in fact equivalent to requiring that an algorithm exhibits *no-regret*, generally defined as follows, against all opponents.

$$\forall \epsilon > 0, \left(lim_{t \to \inf} \left[\frac{1}{t} \max_{a_j \in A_i} r_i^t(a_j, s_i | s_{-i}) \right] < \epsilon \right)$$

In both game theory and artificial intelligence, a large number of algorithms have been shown to satisfy universal consistency or no-regret requirements.

Recommended Reading

Requisite background in game theory can be obtained from the many introductory texts, and most compactly from Leyton-Brown and Shoham (2008). Game theoretic work on multiagent learning is covered in Fudenberg and Levine (1998) and Young (2004). An expanded discussion of the problems addressed under the header of MAL can be found in Shoham, Powers, and Grenager (2007), and the responses to it in Vohra and Wellman (2007). Discussion of MAL algorithms, both traditional and more novel ones, can be found in the above references, as well as in Greenwald and Littman (2007).

Fudenberg, D., & Levine, D. (1998). *The theory of learning in games*. Cambridge: MIT Press.

Greenwald, A., & Littman, M. L. (Eds.). (2007). Special issue on learning and computational game theory. *Machine Learning, 67*(1–2).

Leyton-Brown, K., & Shoham, Y. (2008). *Essentials of game theory*. San Rafael, CA: Morgan and Claypool.

Shoham, Y., Powers, W. R., & Grenager, T. (2007). If multiagent learning is the answer, what is the question? *Artificial Intelligence, 171*(1), 365–377. Special issue on foundations of multiagent learning.

Vohra, R., & Wellman, M. P. (Eds.). (2007). Special issue on foundations of multiagent learning. *Artificial Intelligence, 171*(1).

Young, H. P. (2004). *Strategic learning and its limits*. Oxford: Oxford University Press.

Multi-Armed Bandit

►*k*-Armed Bandit

Multi-Armed Bandit Problem

►*k*-Armed Bandit

MultiBoosting

Geoffrey I. Webb
Monash University,
Victoria, Australia

Definition

MultiBoosting (Webb, 2000) is an approach to ►multi-strategy ensemble learning that combines features of ►AdaBoost and ►Bagging. The insight underlying MultiBoosting is that the primary effect of AdaBoost is ►bias reduction, while the primary effect of bagging is ►variance reduction. By combining the two techniques, it is possible to obtain both bias and variance reduction, the cumulative effect often being a greater reduction in error than can be obtained with the equivalent amount of computation by either AdaBoost or Bagging alone. Viewed from another perspective, as the size of an ensemble formed by either AdaBoost or Bagging is increased, each successive addition to the ensemble has decreasing effect. Thus, if the benefit of the first few applications of AdaBoost can be combined with

the benefit of the first few applications of Bagging, the combined benefit may be greater than simply increasing the number of applications of one or the other.

Algorithm

MultiBoosting operates by dividing the ensemble of classifiers that is to be created into a number of subcommittees. Each of these subcommittees is formed by Wagging (Baner & Kohavi, 1999), a variant of Bagging that utilizes weighted instances and, hence, is more readily integrated with AdaBoost. The ensemble is formed by applying AdaBoost to these subcommittees. The resulting algorithm is presented in Table 1. The learned ensemble classifier is C, and the tth member of the ensemble is C_t. Each S_t is a vector of n weighted training objects whose weights always sum to n. The weights change from turn to turn (the turns indicated by the subscript t). The base training algorithm *BaseLearn* should more heavily penalize errors on training instances with higher weights. ϵ_t is the weighted error

MultiBoosting. Table 1 MultiBoost Algorithm

MultiBoost

input:

- S_0, a sequence of m labeled examples $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ with labels $y_i \in Y$.
- base learning algorithm *BaseLearn*.
- integer T specifying the number of iterations.
- vector of integers I_i specifying the iteration at which each subcommittee $i \geq 1$ should terminate.

1. $S_1 = S_0$ with instance weights assigned to be 1.
2. set $k = 1$.
3. For $t = 1$ to T
4. If $I_k = t$ then
5. reweight S_t.
6. increment k.
7. $C_t = BaseLearn(S')$.
8. $\epsilon_t = \dfrac{\sum_{x_j \in S_t : C_t(x_j) \neq y_j} weight(x_j)}{m}$.
9. if $\epsilon_t > 0.5$ then
10. reweight S_t.
11. increment k.
12. go to 7.
13. otherwise if $\epsilon_t = 0$ then
14. set β_t to 10^{-10}.
15. reweight S_t.
16. increment k.
17. otherwise,
18. $\beta_t = \dfrac{\epsilon_t}{(1 - \epsilon_t)}$.
19. $S_{t+1} = S_t$.
20. For each $x_j \in S_{t+1}$,
21. divide $weight(x_j)$ by $2\epsilon_t$ if $C_t(x_j) \neq y_j$ and $2(1 - \epsilon_t)$ otherwise.
22. if $weight(x_j) < 10^{-8}$, set $weight(x_j)$ to 10^{-8}.

Output the final classifier: $C^*(x) = \underset{y \in Y}{argmax} \sum_{t : C_t(x) = y} log \dfrac{1}{\beta_t}$.

of C_t on S_i. β_t is a weight assigned to the tth classifier, C_t. The operation rewieght S_t sets the weights of the objects in S_t to random values drawn from the continuous Poisson distribution and then standardizes them to sum to n. The code set with a grey background is the code added to AdaBoost in order to create MultiBoost.

Cross References

►AdaBoost
►Bagging
►Ensemble Learning
►Multistrategy Ensemble Learning

Recommended Reading

Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning, 36*(1), 105–139.

Webb, G. I. (2000). MultiBoosting: A technique for combining boosting and wagging. *Machine Learning, 40*(2), 159–196.

Multi-Criteria Optimization

►Multi-Objective Optimization

Multi-Instance Learning

Soumya Ray,[1] Stephen Scott,[2] Hendrik Blockeel[3]
[1]Case Western Reserve University, Cleveland, OH, USA
[2]University of Nebraska, Lincoln, NE, USA
[3]K. U. Leuven, Heverlee, Belgium

Synonyms

Multiple-instance learning

Definition

Multiple-Instance (MI) learning is an extension of the standard supervised learning setting. In standard supervised learning, the input consists of a set of labeled instances each described by an attribute vector. The learner then induces a concept that relates the label of an instance to its attributes. In MI learning, the input consists of labeled examples (called "bags") consisting of *multisets* of instances, each described by an attribute vector, and there are constraints that relate the label of each bag to the unknown labels of each instance. The MI learner then induces a concept that relates the label of a bag to the attributes describing the instances in it. This setting contains supervised learning as a special case: if each bag contains exactly one instance, it reduces to a standard supervised learning problem.

Motivation and Background

The MI setting was introduced by Dietterich, Lathrop, and Lozano-Perez (1997) in the context of drug activity prediction. Drugs are typically molecules that fulfill some desired function by binding to a target. If we wish to learn the characteristics responsible for binding, a possible representation of the problem is to represent each molecule as a set of low energy shapes or *conformations*, and describe each conformation using a set of attributes. Each such bag of conformations is given a label corresponding to whether the molecule is active or inactive. To learn a classification model, an algorithm assumes that every instance in a bag labeled negative is actually negative, whereas at least one instance in a bag labeled positive is actually positive with respect to the underlying concept.

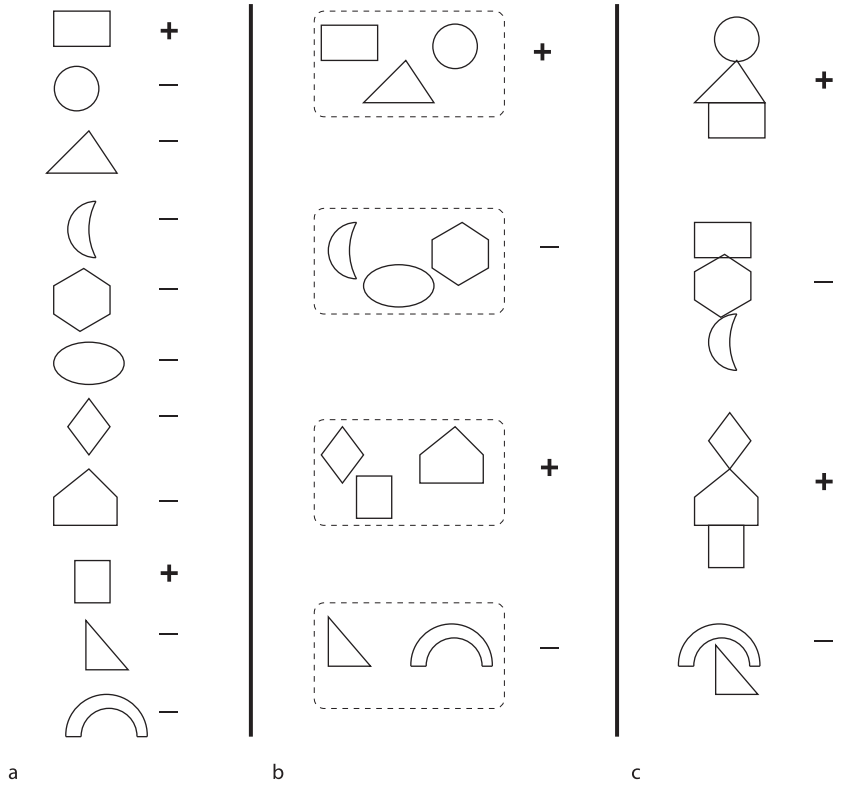
From a theoretical viewpoint, MI learning occupies an intermediate position between standard propositional supervised learning and first-order relational learning. Supervised learning is a special case of MI learning, while MI learning is a special case of first-order learning. It has been argued that the MI setting is a key transition between standard supervised and relational learning DeRaedt (1998). At the same time, theoretical results exist that show that, under certain assumptions, certain concept classes that are probably approximately correct (PAC)-learnable (see PAC Learning) in a supervised setting remain PAC-learnable in an MI setting. Thus, the MI setting is able to leverage some of the rich representational power of relational learners while not sacrificing the efficiency of propositional learners. Figure 1 illustrates the relationships between standard supervised learning, MI learning, and relational learning.

Since its introduction, a wide variety of tasks have been formulated as MI learning problems. Many

new algorithms have been developed, and well-known supervised learning algorithms extended, to learn MI concepts. A great deal of work has also been done to understand what kinds of concepts can and cannot be learned efficiently in this setting. In the following sections, we discuss the theory, methods, and applications of MI learning in more detail.

Structure of the Problem

The general MI classification task in shown in Fig. 2. The MI regression task is defined analogously by substituting a real-valued response for the classification label. In this case, the constraint used by the learning algorithm is that the response of any bag is equal to the



Multi-Instance Learning. Figure 1. The relationship between supervised, multiple-instance (MI), and relational learning. (a) In supervised learning, each example (geometric figure) is labeled. A possible concept that explains the example labels shown is "the figure is a rectangle." (b) In MI learning, bags of examples are labeled. A possible concept that explains the bag labels shown is "the bag contains at least one figure that is a rectangle." (c) In relational learning, objects of arbitrary structure are labeled. A possible concept that explains the object labels shown is "the object is a stack of three figures and the bottom figure is a rectangle"

Given: A set of bags $\{B_1, ...B_n\}$ each with label $\ell_i \in \{0, 1\}$. Each B_i is a multiset of n_i instances, $B_i = \{B_{i1}, \ldots, B_{in_i}\}$.

Constraints: There exists a concept c such that:

- For every B_i with $\ell_i = 1$, $c(B_{ij}) = 1$ for at least one j, and
- For every B_i with $\ell_i = 0$, $c(B_{ij}) = 0$ for all j.

Do: Learn a concept that maps a bag B_i to its label ℓ_i.

Multi-Instance Learning. Figure 2. Statement of the multiple-instance classification problem

response of at least one of the instances in it, for example, it could be equal to the largest response over all the instances.

Notice the following problem characteristics:

- The number of instances in each bag can vary independently of other bags. This implies in particular that an MI algorithm must be able to handle bags with as few as one instance (this is a supervised learning setting) to bags with large numbers of instances.
- The number of instances in any positive bag that are "truly positive" could be many more than one – in fact, the definition does not rule out the case where *all* instances in a positive bag are "truly positive."
- The problem definition does not specify how the instances in any bag are related to each other.

Theory and Methods

In this section we discuss some of the key algorithms and theoretical results in MI learning. We first discuss the methods and results for MI classification. Then we discuss the work on MI regression.

Multiple-Instance Classification

Axis-Parallel Rectangles (APRs) are a concept class that early work in MI classification focused on. These generative concepts specify upper and lower bounds for all numeric attributes describing each instance. An APR is said to "cover" an instance if the instance lies within it. An APR covers a bag if it covers at least one instance within it. The learning algorithm tries to find an APR such that it covers all positive bags and does not cover any negative bags.

An algorithm called "iterated-discrimination" was proposed by Dietterich et al. (1997) to learn APRs from MI data. This algorithm has two phases. In the first phase, it iteratively chooses a set of "relevant" attributes and grows an APR using this set. This phase results in the construction of a very "tight" APR that covers just positive bags. In the second phase, the algorithm expands this APR so that with high probability a new positive instance will fall within the APR. The key steps of the algorithm are outlined below. Note that initially, all attributes are considered to be "relevant."

The algorithm starts by choosing a random instance in a positive bag. Let us call this instance I_1. The smallest APR covering this instance is a point. The algorithm then expands this APR by finding the smallest APR that covers any instance from a yet uncovered positive bag; call the newly covered instance I_2. This process is continued, identifying new instances $I_3, \ldots, I_k$, until all positive bags are covered. At each step, the APR is "backfitted" in a way that is reminiscent of the later Expectation-Maximization (EM) approaches: each earlier choice is revisited, and I_j is replaced with an instance from the same bag that minimizes the current APR (which may or may not be the same as the one that minimized it at step j).

This process yields an APR that imposes maximally tight bounds on all attributes and covers all positive bags. Based on this APR, a new set of "relevant" attributes is selected as follows. An attribute's relevance is determined by how strongly it discriminates against negative instances, i.e., given the current APR bounds, how many negative instances the attribute excludes. Features are then chosen iteratively and greedily according to how relevant they are until all negative instances have been excluded. This yields a subset of (presumably relevant) attributes. The APR growth procedure in the previous paragraph is then repeated, with the size of an APR redefined as its size along relevant attributes only. The APR growth and attribute selection phases are repeated until the process converges.

The APR thus constructed may still be too tight, as it fits narrowly around the positive bags in the dataset. In the second phase of the algorithm, the APR bounds are further expanded using a kernel density estimate approach. Here, a probability distribution is constructed for each relevant attribute using Gaussian distributions centered at each instance in a positive bag. Then, the bounds on that attribute are adjusted so that with high probability, any positive instance will lie within the expanded APR.

Theoretical analyses of APR concepts have been performed along with the empirical approach, using Valiant's "probably approximately correct" (PAC) learning model (Valiant, 1984). In early work (Long & Tan, 1998), it was shown that if each instance was drawn according to a fixed, unknown product distribution over the rational numbers, independently from every other instance, then an algorithm could PAC-learn APRs. Later, this result was improved in two ways (Auer, Long, & Srinivasan, 1998). First, the restriction that the

individual instances in each bag come from a product distribution was removed. Instead, each instance is generated by an arbitrary probability distribution (though each instance in a bag is still generated independently and identically distributed (iid) according to that one distribution). Second, the time and sample complexities for PAC-learning APRs were improved. Specifically, the algorithm described in this work PAC-learns APRs in

$$O\left(\frac{d^3n^2}{\epsilon^2}\log\frac{nd\log(1/\delta)}{\epsilon}\log\frac{d}{\delta}\right)$$

using

$$O\left(\frac{d^2n^2}{\epsilon^2}\log\frac{d}{\delta}\right)$$

time-labeled training bags. Here, d is the dimension of each instance, n is the (largest) number of instances per training bag, and ϵ and δ are parameters to the algorithm. A variant of this algorithm was empirically evaluated and found to be successful (Auer, 1997).

Diverse Density (Maron, 1998; Maron & Lozano-Pérez, 1998) is a probabilistic generative framework for MI classification. The idea behind this framework is that, given a set of positive and negative bags, we wish to learn a concept that is "close" to at least one instance from each positive bag, while remaining "far" from every instance in every negative bag. Thus, the concept must describe a region of instance space that is "dense" in instances from positive bags, and is also "diverse" in that it describes every positive bag. More formally, let

$$DD(t) = \frac{1}{Z}\left(\prod_i \Pr(t|B_i^+)\prod_i \Pr(t|B_i^-)\right),$$

where t is a candidate concept, B_i^+ represents the ith positive bag, and B_i^- represents the ith negative bag. We seek a concept that maximizes $DD(t)$. The concept generates the instances of a bag, rather than the bag itself. To score a concept with respect to a bag, we combine t's probabilities for instances using a function based on noisy-OR Pearl (1998):

$$\Pr(t|B_i^+) \propto (1 - \prod_j (1 - \Pr(B_{ij}^+ \in t))) \quad (1)$$

$$\Pr(t|B_i^-) \propto \prod_j (1 - \Pr(B_{ij}^- \in t)) \quad (2)$$

Here, the instances B_{ij}^+ and B_{ij}^- belonging to t are the "causes" of the "event" that "t is the target." The concept class investigated by Maron (1998) is the class of generative Gaussian models, which are parameterized by the mean μ and a "scale" $s = \frac{1}{2\sigma^2}$:

$$\Pr(B_{ij} \in t) \propto e^{-\sum_k (s_k (B_{ijk} - \mu_k)^2)},$$

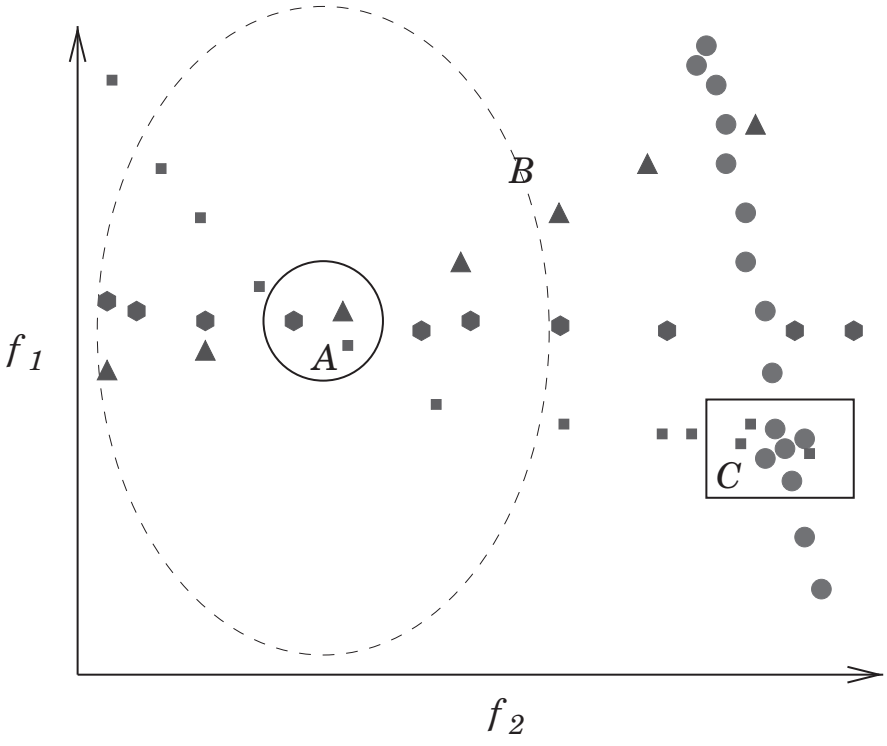
where k ranges over attributes. Figure 3 illustrates a concept that Diverse Density might learn when applied to an MI dataset.

Diverse Density with k disjuncts is a variant of Diverse Density that has also been investigated (Maron, 1998). This is a class of disjunctive Gaussian concepts, where the probability of an instance belonging to a concept is given by the maximum probability of belonging to any of the disjuncts.

EM-DD (Zhang & Goldman, 2001) is an example of a class of algorithms that try to identify the "cause" of a bag's label using EM. These algorithms sometimes assume that there is a single instance in each bag that is responsible for the bag's label (though variants using "soft EM" are possible). The key idea behind this approach is as follows: from each positive bag, we take a random instance and assume that this instance is the relevant one. We learn a hypothesis from these relevant instances and all negative bags. Next, for each positive bag, we replace the current relevant instance by the instance most consistent with the learned hypothesis (which will initially not be the chosen instance in general). We then relearn the hypothesis with these new instances. This process is continued until the set of chosen instances does not change (or alternatively, the objective function of the classifier reaches a fixed point). This procedure has the advantage of being computationally efficient, since the learning algorithm only uses one instance from each positive bag. This approach has also been used in MI regression described later.

"Upgraded" supervised learning algorithms can be used in a MI setting by suitably modifying their objective functions. Below, we summarize some of the algorithms that have been derived in this way.

1. ▶*Decision Tree induction* algorithms have been adapted to the MI setting (Blockeel, Page, & Srinivasan, 2005). The standard algorithm measures



Multi-Instance Learning. Figure 3. An illustration of the concept that Diverse Density searches for on a simple MI dataset with three positive bags and one negative bag, where each instance (represented by the geometric figures) is described by two attributes, f_1 and f_2. Each type of figure represents one bag, i.e., all triangles belong to one bag, all circles belong to a second bag, and so forth. The bag containing the red circles is negative, while the other bags are positive. Region *C* is a region of high density, because several instances belong to that region. Region *A* is a region of high "Diverse Density," because several instances *from different positive bags* belong to that region, and no instances from negative bags are nearby. Region *B* shows a concept that might be learned if the learning algorithm assumed that all instances in every positive bag are positive. Figure adapted from Maron (1998)

the quality of a split on an attribute by considering the class label distribution in the child nodes produced. In the MI case, this distribution is uncertain, because the true instance labels in positive bags are unknown. However, some rules have been identified that lead to empirically good MI trees: (1) use an asymmetric heuristic that favors early creation of pure positive (rather than negative) leaves, (2) once a positive leaf has been created, remove all other instances of the bags covered by this leaf; (3) abandon the depth-first or breadth-first order in which nodes are usually split, adopting a best-first strategy instead (indeed, because of (2), the result of tree learning is now sensitive to the order in which the nodes are split).

2. ►*Artificial Neural Networks* have been adapted to the MI setting by representing the bag classifier as a network that combines several copies of a smaller network, which represents the instance classifier, with a smooth approximation of the *max* combining function (Ramon & DeRaedt, 2000). Weight update rules for a backpropagation algorithm working on this network have been derived. Later work on MI neural networks has been performed independently by others (Zhou & Zhang, 2002).
3. ►*Logistic Regression* has been adapted to the MI setting by using it as an instance-based classifier and combining the instance-level probabilities using functions like softmax (Ray & Craven, 2005) and

arithmetic and geometric averages (Xu & Frank, 2004).

4. The ►*k-Nearest Neighbor* algorithm has been adapted to the MI setting by using set-based distance metrics, such as variants based on the Hausdorff distance. However, this alone does not solve the problem – it is possible for a positive bag to be mistakenly classified negative if it contains a "true negative" instance that happens to be much closer to negative instances in other negative bags. To solve this, a "Citation-kNN" (Wang & Zucker, 2000) approach has been proposed that also considers, for each bag B, the labels of those bags for which B is a nearest neighbor.
5. ►*Support Vector Machines* have been adapted to the MI setting in several ways. In one method, the constraints in the quadratic program for SVMs is modified to account for the fact that certain instance labels are unknown but have constraints relating them (Andrews, Tsochantaridis, & Hofmann, 2003). In another method, new kernels are designed for MI data by modifying standard supervised SVM kernels (Gartner, Flach, Kowalczyk, & Smola, 2002) or designing new kernels (Tao, Scott, & Vinodchandran, 2004). The modification allows these MI kernels to distinguish between positive and negative bags if the supervised kernel could distinguish between ("true") positive and negative instances.
6. ►*Rule learning algorithms* have been adapted to the MI setting in two ways. One method has investigated upgrading a supervised rule-learner, the RIPPER system (Cohen, 1995), to the MI setting by modifying its objective function to account for bags and addressing several issues that resulted. Another method has investigated using general purpose relational algorithms, such as FOIL (Quinlan, 1990) and TILDE (Blockeel & De Raedt, 1998), and providing them with an appropriate ►inductive bias so that they learn the MI concepts. Further, it has been observed that techniques from MI learning can also be used inside relational learning algorithms (Alphonse & Matwin, 2002).

A large-scale empirical analysis of several such propositional supervised learning algorithms and their MI counterparts has been performed (Ray & Craven, 2005). This analysis concludes that (1) no single MI algorithm works well across all problems. Thus, different inductive biases are suited to different problems, (2) some MI algorithms consistently perform better than their supervised counterparts but others do not (hence for these biases there seems room for improvement), and (3) assigning a larger weight to false positives than to false negatives is a simple but effective method to adapt supervised learning algorithms to the MI setting. It was also observed that the advantages of MI learners may be more pronounced if they would be evaluated on the task of labeling individual instances rather than bags.

Along with "upgrading" supervised learning algorithms, a *theoretical analysis of supervised learners* learning with MI data has been carried out (Blum & Kalai, 1998). In particular, the MI problem has been related to the problem of learning in the presence of classification noise (i.e., each training example's label is flipped with some probability $< 1/2$). This implies that any concept class that is PAC-learnable in the presence of such noise is also learnable in the MI learning model when each instance of a bag is drawn iid. Since many concept classes are learnable under this noise assumption (using e.g., *statistical queries* Kearns, 1998), Blum and Kalai's result implies PAC learnability of many concept classes. Further, they improved on previous learnability results (Auer et al., 1998) by reducing the number of training bags required for PAC learning by about a factor of n with only an increase in time complexity of about $\log n/\epsilon$.

Besides these positive results, a *negative learnability result* describing when it is hard to learn concepts from MI data is also known (Auer et al., 1998). Specifically, if the instances of each bag are allowed collectively to be generated according to an arbitrary distribution, learning from MI examples is as hard as PAC-learning disjunctive normal form (DNF) formulas from single-instance examples, which is an open problem in learning theory that is believed to be hard. Further, it has been showed that if an efficient algorithm exists for the non-iid case that outputs as its hypothesis an axis-parallel rectangle, then NP = RP (Randomized Polynomial time, see e.g., Papadimitriou, 1994), which is very unlikely.

Learning from structured MI data has received some attention (McGovern & Jensen, 2003). In this work,

each instance is a graph, and a bag is a set of graphs (e.g., a bag could consist of certain subgraphs of a larger graph). To learn the concepts in this structured space, the authors use a modified form of the Diverse Density algorithm discussed above. As before, the concept being searched for is a point (which corresponds to a graph in this case). The main modification is the use of the size of the maximal common subgraph to estimate the probability of a concept – i.e., the probability of a concept given a bag is estimated as proportional to the size of the maximal common subgraph between the concept and any instance in the bag.

Multiple-Instance Regression

Regression problems in an MI setting have received less attention than the classification problem. Two key directions have been explored in this setting. One direction extends the well-known standard ►linear regression method to the MI setting. The other direction considers extending various MI classification methods to a regression setting.

In *MI Linear Regression* (Ray & Page, 2001) (referred to as multiple-instance regression in the cited work), it is assumed that the hypothesis underlying the data is a linear model with Gaussian noise on the value of the dependent variable (which is the response). Further, it is assumed that it is sufficient to model one instance from each bag, i.e., that there is some *primary* instance which is responsible for the real-valued label. Ideally, one would like to find a hyperplane that minimizes the squared error with respect to these primary instances. However, these instances are unknown during training. The authors conjecture that, given enough data, a good approximation to the ideal is given by the "best-fit" hyperplane, defined as the hyperplane that minimizes the training set squared error by fitting one instance from each bag such that the response of the fitted instance most closely matches the bag response. This conjecture will be true if the nonprimary instances are not a better fit to a hyperplane than the primary instances. However, exactly finding the "best-fit" hyperplane is intractable. It is shown that the decision problem "Is there a hyperplane which perfectly fits one instance from each bag?" is *NP*-complete for arbitrary numbers of bags, attributes, and at most three instances per bag. Thus, the authors propose an approximation algorithm which iterates between choosing instances and learning linear regression models that best fit them, similar to the EM-DD algorithm described earlier.

Another direction has explored *extending MI classification algorithms* to the regression setting. This approach (Dooly, Zhang, Goldman, & Amar, 2002) uses algorithms like Citation-kNN and Diverse Density to learn real-valued concepts. To predict a real value, the approach uses the average of the nearest neighbor responses or interprets the Gaussian "probability" as a real number for Diverse Density.

Recent work has analyzed the Diverse Density-based regression in the *online* model (Angluin, 1988; Littlestone, 1988) (see ►online learning). In the online model, learning proceeds in *trials*, where in each trial a single example is selected adversarially and given to the learner for classification. After the learner predicts a label, the true label is revealed and the learner incurs a *loss* based on whether its prediction was correct. The goal of the online learner is to minimize the loss over all trials. Online learning is harder than PAC learning in that there are some PAC-learnable concept classes that are not online learnable.

In the regression setting above (Dooly, Goldman, & Kwek, 2006), there is a point concept, and the label of each bag is a function of the distance between the concept and the point in the bag closest to the target. It is shown that similar to Auer et al.'s lower bound, learning in this setting using labeled bags alone is as hard as learning DNF. They then define an *MI membership query* (MI-MQ) in which an adversary defines a bag $B = \{p_1, \ldots, p_n\}$ and the learner is allowed to ask an oracle for the label of bag $B + \vec{v} = \{p_1 + \vec{v}, \ldots, p_n + \vec{v}\}$ for any d-dimensional vector $\vec{v}$. Their algorithm then uses this MI-MQ oracle to online learn a real-valued MI concept in time $O(dn^2)$.

Applications

In this section, we describe domains where MI learning problems have been formulated.

Drug activity was the motivating application for the MI representation (Dietterich et al., 1997). Drugs are typically molecules that fulfill some desired function by binding to a target. In this domain, we wish to predict how strongly a given molecule will bind to a target. Each molecule is a three-dimensional entity and

takes on multiple shapes or *conformations* in solution. We know that for every molecule showing activity, at least one of its low energy conformations possesses the right shape for interacting with the target. Similarly, if the molecule does not show drug-like activity, none of its conformations possess the right shape for interaction. Thus, each molecule is represented as a bag, where each instance is a low energy conformation of the molecule. A well-known example from this domain is the MUSK dataset. The positive class in this data consists of molecules that smell "musky." This dataset has two variants, MUSK1 and MUSK2, both with similar numbers of bags, with MUSK2 having many more instances per bag.

Content-Based Image Retrieval is another domain where the MI representation has been used (Maron & Lozano-Pérez, 1998; Zhang, Yu, Goldman, & Fritts, 2002). In this domain, the task is to find images that contain objects of interest, such as tigers, in a database of images. An image is represented by a bag. An instance in a bag corresponds to a segment in the image, obtained by some segmentation technique. The underlying assumption is that the object of interest is contained in (at least) one segment of the image. For example, if we are trying to find images of mountains in a database, it is reasonable to expect most images of mountains to have certain distinctive segments characteristic of mountains. An MI learning algorithm should be able to use the segmented images to learn a concept that represents the shape of a mountain and use the learned concept to collect images of mountains from the database.

The *identification of protein families* has been framed as an MI problem (Tao et al., 2004). The objective in that work is to classify given protein sequences according to whether they belong to the family of thioredoxin-fold proteins. The given proteins are first aligned with respect to a motif that is known to be conserved in the members of the family. Each aligned protein is represented by a bag. A bag is labeled positive if the protein belongs to the family, and negative otherwise. An instance in a bag corresponds to a position in a fixed length sequence around the conserved motif. Each position is described by a vector of attributes; each attribute describes the properties of the amino acid at that position, and is smoothed using the same properties from its neighbors.

Text Categorization is another domain that has used the MI representation (Andrews et al., 2003; Ray & Craven 2005). In this domain, the task is to classify a document as belonging to a certain category or not. Often, whether the document belongs to the specified category is the function of a few passages in the document. These passages are however not labeled with the category information. Thus, a document could be represented as a set of passages. We assume that each positive document (i.e., that belongs to the specified category) has at least one passage that contains words that indicate category membership. On the other hand, a negative document (that does not belong to the category) has no passage that contain words indicating category membership. This formulation has been used to classify whether MEDLINE documents should be annotated with specific MeSH terms (Andrews et al.) and to determine if specific documents should be annotated with terms from the Gene Ontology (Ray & Craven, 2005).

Time-series data from the hard drives have been used to define an MI problem (Murray, Hughes, & Kreutz-Delgado, 2005). The task here is to distinguish drives that fail from others. Each hard drive is a bag. Each instance in the bag is a fixed-size window over timepoints when the drive's state was measured using certain attributes. In the training set, each drive is labeled according to whether it failed during a window of observation. An interesting aspect to prediction in this setting is that it is done online, i.e., the algorithm learns a classifier for instances, which is applied to each instance as it becomes available in time. The authors learn a naïve Bayes model using an EM-based approach to solve this problem.

Discovering useful subgoals in reinforcement learning has been formulated as an MI problem (McGovern & Barto, 2001). Imagine that a robot has to get from one room to another by passing through a connecting door. If the robot knew of the existence of the door, it could decompose the problem into two simpler subproblems to be solved separately: getting from the initial location in the first room to the door, and then getting from the door to its destination. How could the robot discover such a "useful subgoal?" One approach formulates this as an MI problem. Each trajectory of the robot, where the robot starts at the source and then moves for some number of time steps, is considered to be a bag. An

instance in a bag is a state of the world, that records observations such as, "is the robot's current location a door?" Trajectories that reach the destination are positive, while those that do not are negative. Given this data, we can learn a classifier that predicts which states are more likely to be seen on successful trajectories than on unsuccessful ones. These states are taken to be useful subgoals. In the previous example, the MI algorithm could learn that the state "location is a door" is a useful subgoal, since it appears on all successful trajectories, but infrequently on unsuccessful ones.

Future Directions

MI learning remains an active research area. One direction that is being explored relaxes the "Constraints" in Fig. 2 in different ways (Tao et al., 2004; Weidmann, Frank, & Pfahringer 2003). For example, one could consider constraints where at least a certain number (or fraction) of instances have to be positive for a bag to be labeled positive. Similarly, it may be the case that a bag is labeled positive only if it does not contain a specific instance. Such relaxations are often studied as "generalized multiple-instance learning."

One such generalization of MI learning has been formally studied under the name "geometric patterns." In this setting, the target concept consists of a collection of APRs, and a bag is labeled positive if and only if (1) each of its points lies in a target APR, and (2) every target APR contains a point. Noise-tolerant PAC algorithms (Goldman & Scott, 1999) and online algorithms (Goldman, Kwek, & Scott, 2001) have been presented for such concept classes. These algorithms make no assumptions on the distribution used to generate the bags (e.g., instances might not be generated by an iid process). This does not violate Auer et al.'s lower bound since these algorithms do not scale with the dimension of the input space.

Another recent direction explores the connections between MI and semi-supervised learnings. Semi-supervised learning generally refers to learning from a setting where some instance labels are unknown. MI learning can be viewed as one example of this setting. Exploiting this connection between MI learning and other methods for semi-supervised learning, recent work (Rahmani & Goldman, 2006) proposes an approach where an MI problem is transformed into a semi-supervised learning problem. An advantage of the approach is that it automatically also takes into account unlabeled bags.

Cross References

- ▶Artificial Neural Network
- ▶Attribute
- ▶Classification
- ▶Data Set
- ▶Decision Trees
- ▶Expectation-Maximization
- ▶First-Order Rule
- ▶Gaussian Distribution
- ▶Inductive Logic Programming
- ▶Kernel Methods
- ▶Linear Regression
- ▶Nearest Neighbor
- ▶Noise
- ▶On-Line Learning
- ▶PAC Learning
- ▶Relational Learning
- ▶Supervised Learning

Recommended Reading

Alphonse, E., & Matwin, S. (2002). Feature subset selection and inductive logic programming. In Proceedings of the 19th International Conference on Machine Learning (pp. 11–18). Morgan Kaufmann, San Francisco, USA.

Andrews, S., Tsochantaridis, I., & Hofmann, T. (2003). Support vector machines for multiple-instance learning. In S. Becker, S. Thrun, & K. Obermayer, (Eds.), *Advances in neural information processing systems*. (Vol. 15, pp. 561–568). Cambridge, MA: MIT Press.

Angluin, D. (1988). Queries and concept learning. *Machine Learning, 2*(4), 319–342.

Auer, P. (1997). On learning from multi-instance examples: Empirical evaluation of a theoretical approach. In *Proceeding of 14th international conference on machine learning* (pp. 21–29). San Francisco: Morgan Kaufmann.

Auer, P., Long, P. M., & Srinivasan, A. (1998). Approximating hyper-rectangles: Learning and pseudorandom sets. *Journal of Computer and System Sciences, 57*(3), 376–388.

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first order logical decision trees. *Artificial Intelligence, 101*(1–2), 285–297.

Blockeel, H., Page, D., & Srinivasan, A. (2005). Multi-instance tree learning. In *Proceedings of 22nd international conference on machine learning* (pp. 57–64). Bonn, Germany.

Blum, A., & Kalai, A. (1998). A note on learning from multiple-instance examples. *Machine Learning Journal, 30*(1), 23–29.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the 12th international conference on machine learning*. San Francisco: Morgan Kaufmann.

DeRaedt, L. (1998). Attribute-value learning versus inductive logic programming: The missing links. In *Proceedings of the eighth international conference on inductive logic programming* (pp. 1–8). New York: Springer.

Dietterich, T., Lathrop, R., & Lozano-Perez, T. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence, 89*(1–2), 31–71.

Dooly, D. R., Goldman, S. A., & Kwek, S. S. (2006). Real-valued multiple-instance learning with queries. *Journal of Computer and System Sciences, 72*(1), 1–15.

Dooly, D. R., Zhang, Q., Goldman, S. A., & Amar, R. A. (2002). Multiple-instance learning of real-valued data. *Journal of Machine Learning Research, 3*, 651–678.

Gartner, T., Flach, P. A., Kowalczyk, A., & Smola, A. J. (2002). Multi-instance kernels. In C. Sammut, & A. Hoffmann, (Eds.), *Proceedings of the 19th international conference on machine learning* (pp. 179–186). San Francisco: Morgan Kaufmann.

Goldman, S. A., Kwek, S. K., & Scott, S. D. (2001). Agnostic learning of geometric patterns. *Journal of Computer and System Sciences, 6*(1), 123–151.

Goldman, S. A., & Scott, S. D. (1999). A theoretical and empirical study of a noise-tolerant algorithm to learn geometric patterns. *Machine Learning, 37*(1), 5–49.

Kearns, M. (1998). Efficient noise-tolerant learning from statistical queries. *Journal of the ACM, 45*(6), 983–1006.

Long, P. M., & Tan, L. (1998). PAC learning axis-aligned rectangles with respect to product distributions from multiple-instance examples. *Machine Learning, 30*(1), 7–21.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning, 2*(4), 285–318.

Maron, O. (1998). Learning from ambiguity. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA.

Maron, O., & Lozano-Pérez, T. (1998). A framework for multiple-instance learning. In M. I. Jordan, M. J. Kearns, & S. A. Solla, (Eds.), *Advances in neural information processing systems* (Vol. 10, pp. 570–576). Cambridge, MA: MIT Press.

McGovern, A., & Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the 18th international conference on machine learning* (pp. 361–368). San Francisco: Morgan Kaufmann.

McGovern, A., & Jensen, D. (2003). Identifying predictive structures in relational data using multiple instance learning. In *Proceedings of the 20th international conference on machine learning* (pp. 528–535). Menlo Park, USA: AAAI Press.

Murray, J. F., Hughes, G. F., & Kreutz-Delgado, K. (2005). Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning Research, 6*, 783–816.

Papadimitriou, C. (1994). *Computational complexity*. Boston, MA: Addison-Wesley.

Pearl, J. (1998). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 5*, 239–266.

Rahmani, R., & Goldman, S. A. (2006). MISSL: Multiple-instance semi-supervised learning. In *Proceedings of the 23rd international conference on machine learning* (pp. 705–712). New York, USA: ACM Press.

Ramon, J., & DeRaedt, L. (2000). Multi instance neural networks. In *Proceedings of ICML-2000 workshop on attribute-value and relational learning.*

Ray, S., & Craven, M. (2005). Supervised versus multiple-instance learning: An empirical comparison. In *Proceedings of the 22nd international conference on machine learning* (pp. 697–704). New York: ACM Press.

Ray, S., & Page, D. (2001). Multiple instance regression. In *Proceedings of the 18th international conference on machine learning*. Williamstown, MA: Morgan Kaufmann.

Tao, Q., Scott, S. D., & Vinodchandran, N. V. (2004). SVM-based generalized multiple-instance learning via approximate box counting. In *Proceedings of the 21st international conference on machine learning* (pp. 779–806). San Francisco: Morgan Kaufmann.

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

Wang, J., & Zucker, J. D. (2000). Solving the multiple-instance problem: A lazy learning approach. In *Proceedings of the 17th international conference on machine learning* (pp. 1119–1125). San Francisco: Morgan Kaufmann.

Weidmann, N., Frank, E., & Pfahringer, B. (2003). A two-level learning method for generalized multi-instance problems. In *Proceedings of the European conference on machine learning* (pp. 468–479). Berlin/Heidelberg: Springer.

Xu, X., & Frank, E. (2004). Logistic regression and boosting for labeled bags of instances. In *Proceedings of the Pacific-Asia conference on knowledge discovery and data mining* (pp. 272–281). Sydney, Australia.

Zhang, Q., & Goldman, S. (2001). EM-DD: An improved multiple-instance learning technique. In *Advances in Neural Information Processing Systems* (pp. 1073–1080). MIT Press.

Zhang, Q., Yu, W., Goldman, S., & Fritts, J. (2002). Content-based image retrieval using multiple-instance learning. In *Proceedings of the 19th international conference on machine learning* (pp. 682–689). San Francisco: Morgan Kaufmann.

Zhou, Z. H., & Zhang, M. L. (2002). Neural networks for multi-instance learning. Technical Report, Nanjing University, Nanjing, China.

Multi-Objective Optimization

Synonyms

MOO; Multi-criteria optimization; Vector optimization

Definition

Multi-criteria optimization is concerned with the optimization of a vector of objectives, which can be the subject of a number of constraints or bounds. The goal of multi-objective optimization is usually to find or to approximate the set of Pareto-optimal solutions. A solution is Pareto-optimal if it cannot be improved in one objective without getting worse in another one.

Multiple Classifier Systems

►Ensemble Learning

Multiple-Instance Learning

►Multi-Instance Learning

Multi-Relational Data Mining

Luc De Raedt
Katholieke Universiteit Leuven,
Heverlee, Belgium

Synonyms

Inductive logic programming; Relational learning; Statistical relational learning

Definition

Multi-relational data mining is the subfield of knowledge discovery that is concerned with the mining of multiple tables or relations in a database. This allows it to cope with structured data in the form of complex data that cannot easily be represented using a single table, or an ►attribute as is common in machine learning.

Relevant techniques of multi-relational data mining include those from relational learning, statistical relational learning, and inductive logic programming.

Cross References

►Inductive Logic Programming

Recommended Reading

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

Multistrategy Ensemble Learning

Definition

Every ►ensemble learning strategy might be expected to have unique effects on the base learner. Combining multiple ensemble learning algorithms might hence be expected to provide benefit. For example, ►MultiBoosting combines ►AdaBoost and a variant of ►Bagging, obtaining most of AdaBoost's ►bias reduction coupled with most of Bagging's ►variance reduction. Similarly, ►Random Forests combines Bagging's variance reduction with ►Random Subspaces' bias reduction.

Cross References

►Ensemble Learning
►MultiBoosting
►Random Forests

Recommended Reading

Webb, G. I., & Zheng, Z. (2004). Multistrategy ensemble learning: Reducing error by combining ensemble learning techniques. *IEEE Transactions on Knowledge and Data Engineering, 16*(8), 980–991.

Must-Link Constraint

A pairwise constraint between two items indicating that they should be placed into the same cluster in the final partition.