C

Candidate-Elimination Algorithm

Mitchell's, (1982; 1997) candidate-elimination algorithm performs a bidirectional search in the ▶ hypothesis space. It maintains a set, S, of most specific hypotheses that are consistent with the training data and a set, G, of most general hypotheses consistent with the training data. These two sets form two boundaries on the version space.

Recommended Reading

Mitchell TM (1982) Generalization as search Artif Intell 18(2):203–226
Mitchell TM (1997) Machine learning. McGraw-Hill, New York

Cannot-Link Constraint

A pairwise constraint between two items indicating that they should be placed into different clusters in the final partition.

Cascade Correlation

Thomas R. Shultz[1] and Scott E. Fahlman[2]
[1]McGill University, Montréal, QC, Canada
[2]Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

Cascor; CC

Definition

Cascade–correlation (often abbreviated as "Cascor" or "CC") is a supervised learning algorithm for artificial neural networks. It is related to the back-propagation algorithm ("backprop"). CC differs from backprop in that a CC network begins with no hidden units and then adds units one by one, as needed during learning.

Each new hidden unit is trained to correlate with residual error in the network built so far. When it is added to the network, the new unit is frozen, in the sense that its input weights are fixed. The hidden units form a *cascade*: each

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

new unit receives weighted input from all the original network inputs and from the output of every previously created hidden unit. This cascading creates a network that is as deep as the number of hidden units. Stated another way, the CC algorithm is capable of efficiently creating complex, higher-order nonlinear basis functions – the hidden units – which are then combined to form the desired outputs.

The result is an algorithm that learns complex input/output mappings very fast compared to backprop and that builds a multilayer network structure that is customized for the problem at hand.

Motivation and Background

Cascade–correlation was designed (Fahlman and Lebiere 1990) to address two well-known problems with the popular back-propagation algorithm ("backprop"). First, a backprop user has to guess what network structure – the number of hidden layers and the number of units in each layer – would be best for a given learning problem. If the network is too small or too shallow, it won't solve the problem; if it is too large or too deep, training is very slow, and the network is prone to overfitting the training data. Because there is no reliable way to choose a good structure before training begins, most backprop users have to train many different structures before finding one that is well matched to the task.

Second, even if a backprop user manages to choose a good network structure, training is generally very slow. That is particularly true in networks with many hidden units or with more than one hidden layer. One cause of slow learning in backprop is the use of a uniform learning rate parameter for updating network weights. This problem was addressed with the Quickprop algorithm (Fahlman 1988), an approximation to Newton's method that adapts the learning rate for each weight parameter depending on the first two derivatives of the local error surface. Quickprop improved learning speed, sometimes dramatically, but learning was still too slow in large or deep networks.

Another cause of slow learning in backprop is the "herd effect" (Fahlman and Lebiere 1990). If the solution to a network problem requires, say, 30 hidden units, each of these units must be trained to do a different job – that is, to compute a different nonlinear basis function. Each hidden unit starts with a different and randomly chosen set of input weights; but if the units are all trained at once, they all see the same error signal. There is no central authority telling each unit to do a separate job, so they tend to drift toward the same part of parameter space, forming a herd that moves around together. Eventually, the units may drift apart and begin to differentiate, but there is nothing to compel this, so the process is slow and unreliable. Usually, in selecting an initial topology for a backprop net, it is necessary to include many extra hidden units to increase the odds that each job will be done by some unit.

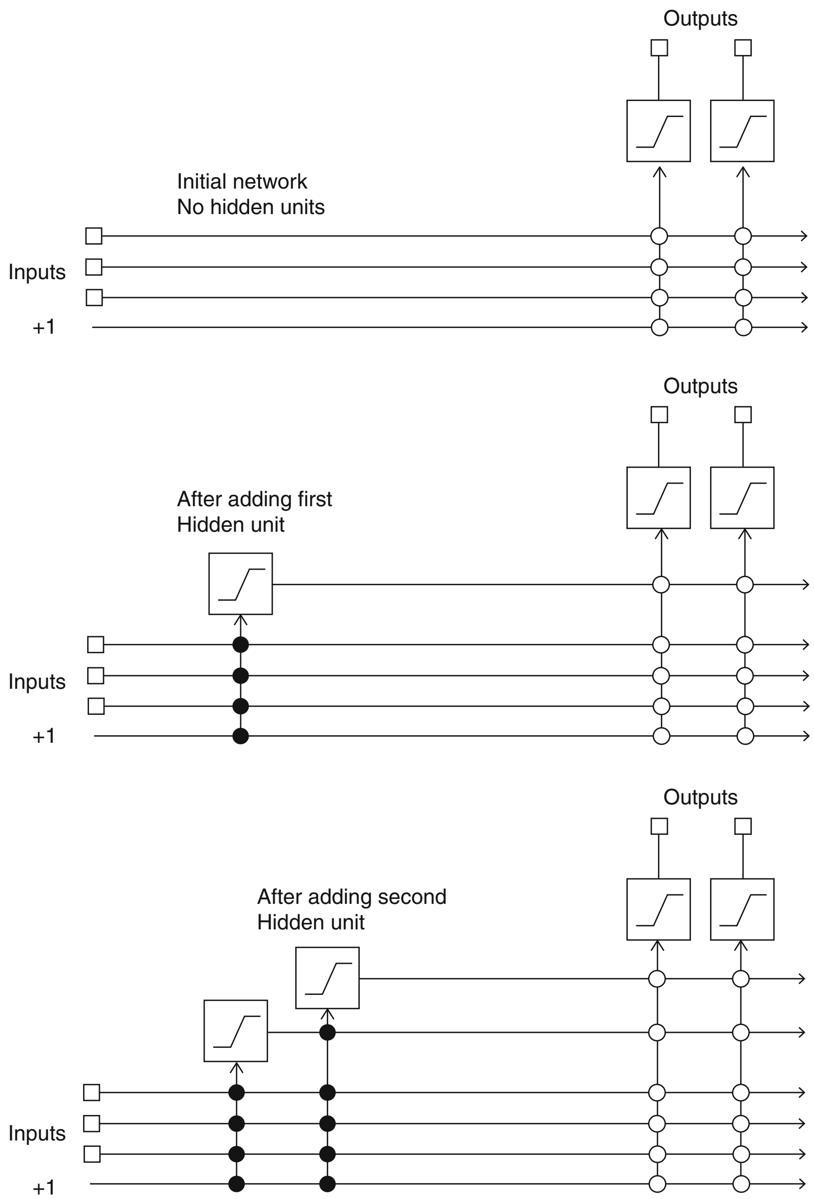
CC addresses this problem by introducing and training hidden units one by one. Each hidden unit sees a strong, clear error gradient, not confused by the simultaneous movement of other hidden units. A new hidden unit can thus move quickly and decisively to a position in parameter space where it can perform a useful function, reducing the residual error. One by one, Cascor-hidden units take up distinct jobs, instead of milling about together competing to do the same job.

Structure of Learning System

The Algorithm

The CC architecture is illustrated in Fig. 1. It begins with some inputs and one or more output units, but no hidden units. The numbers of inputs and outputs are dictated by the problem. As in backprop, the output units generally have a sigmoid activation function, but could alternatively have a linear activation function. Every input is connected to every output unit by a connection with an adjustable weight. There is also a *bias* input, permanently set to $+1$.

Hidden units are added to the network one by one. Each new hidden unit receives a weighted connection from each of the network's original



Cascade Correlation, Fig. 1 The Cascade–correlation (CC) architecture, as new hidden units are added. *Black circles* are frozen connection weights; *white circles* are weights trained during output training phase. The *vertical lines* sum all incoming activation

inputs and also from every existing hidden unit. Each new unit therefore adds a new single-unit layer to the network. This makes it possible to create high-order nonlinear feature detectors, customized for the problem at hand.

As noted, learning begins without hidden units. The direct input–output connections are trained as well as possible over the entire set of training examples, using Quickprop. At some point, this training approaches an asymptote. When no significant error reduction has occurred after a certain number of training cycles, this output phase is terminated and there is a shift to input phase to recruit a new hidden unit, using the unit-creation algorithm to be described. The new unit is added to the net, its input weights are frozen, and all the output weights are once again trained using Quickprop. This cycle repeats until the error is acceptably small, in the sense that all network outputs for all training patterns are within a specified threshold of their target values.

To create a new hidden unit, input phase begins with several *candidate units* that receive trainable input connections from all of the network inputs and from all existing hidden units. The outputs of these candidates are not yet connected to the network. There are a number of passes over the examples of the training set, adjusting the candidate unit's input weights after each pass. The goal of these adjustments, using Quickprop, is to maximize the correlation between each candidate's output and the residual error.

When these correlation measures show no further significant improvement, input phase stops, the best-correlating candidate's input weights are frozen, and that unit is installed in the network. The remaining candidates are discarded and the algorithm then retrains the output weights, making use of this new feature as well as all the old ones. As the new unit's output correlates well with some component of the residual error, its output weights can be quickly adjusted to reduce that component. So after adding each new hidden unit, the network's residual error should be smaller than before.

Using several candidates, each with differently initialized input weights, greatly reduces the chances of installing a bad hidden unit that gets the network stuck in a local optimum far from the global optimum value. All candidates receive the same input signals and see the same residual error for each training pattern. Because they do not interact with one another or affect the network during training, these candidates can be trained in parallel. In a pool of four to eight candidates, there are almost always several high-quality candidates with nearly equal correlation values.

Hidden units continue to be recruited until network error reaches an acceptable level, or until cross-validation signals a stop. Because only a single layer of weights is adjusted at a time, rather than back-propagating an error signal through several layers of shifting units, CC training proceeds very quickly.

Performance

CC is designed to produce a network just large enough to solve the problem and to do so much faster than backprop and related algorithms. In many reported cases that require hidden units, CC learns the desired behavior 10–100 times faster than standard backprop (Fahlman and Lebiere 1990). One striking example of this is the *two-spirals problem*, an artificial benchmark designed to be very difficult for neural networks with sigmoid units. At the time CC was developed, the best known backprop solutions for two spirals required a network with three hidden layers of five units each. CC typically solves this problem with 12 hidden units and has found solutions with as few as nine hidden units. In terms of runtime, CC training was about 50 times faster than standard backprop and 23 times faster than Quickprop used within a static network.

Variants of Cascade Correlation

Flat Cascade Correlation

In standard CC, each new hidden unit receives inputs from every existing unit, so the net becomes one level deeper every time a unit is added. This is a powerful mechanism, creating increasingly complex feature detectors as the network learns. But sometimes this added depth is not required for the problem, creating a very deep network that performs no better than a shallow one. The resulting network might have more weights than are required for the problem, raising concern about overfitting. Another concern was that the cascaded nonlinearity of CC might also compromise generalization. To address these concerns, a flat variant of Cascor adds new recruited units onto a single layer (i.e., cascaded connections are eliminated), limiting the depth of the network and eliminating all cascaded weights between hidden units.

Comparison of flat to standard CC on generalization in particular learning problems yielded inconsistent results, but a more problem–neutral, student–teacher approach found no generalization differences between flat and standard versions of CC (Dandurand et al. 2007). Here, flat and standard student CC networks learned the input–output mappings of other randomly initialized flat and standard CC teacher networks, where task complexity was systematically manipulated. Both standard and flat CC student networks learned and generalized well on problems of varying complexity. In low-complexity tasks, there were no significant performance differences between flat and standard CC student networks. For high-complexity tasks, flat CC student networks required fewer connection weights and learned with less computational cost than did standard CC student networks.

Sibling–Descendant Cascade–Correlation (SDCC)

SDCC (Baluja and Fahlman 1994) provides a more general solution to the problem of network depth. In the candidate pool, there are two kinds of candidate units: *descendant* units that receive inputs from all existing hidden units and *sibling* units that receive the same inputs as the deepest hidden units in the current net. When the time comes to choose a winning candidate, the candidate with the best correlation wins, but there is a slight preference for sibling units. So unless a descendant unit is clearly superior, a sibling unit will be recruited, making the active network larger, but not deeper. In problems where standard CC produces a network with 15 or 20 hidden units and an equal number of layers, SDCC often produces a network with only two or three hidden layers.

Recurrent Cascade–Correlation (RCC)

Standard CC produces a network that maps its *current* inputs to outputs. The network has no memory of recent inputs, so this architecture is not able to learn to recognize a sequence of inputs. In the RCC algorithm, each candidate and hidden unit takes the same inputs as in standard CC, but it also takes an additional input: the unit's own previous output, delayed by one time interval (Fahlman 1991). The weight on this time-delayed input is trained by the same algorithm as all the other inputs.

This delayed loop gives RCC networks a way of remembering past inputs and internal states, so they can learn to recognize sequences of input patterns. In effect, the architecture builds a finite-state machine tailored specifically to recognize the pattern sequences in the training set. For example, an RCC net learned to recognize characters in Morse code.

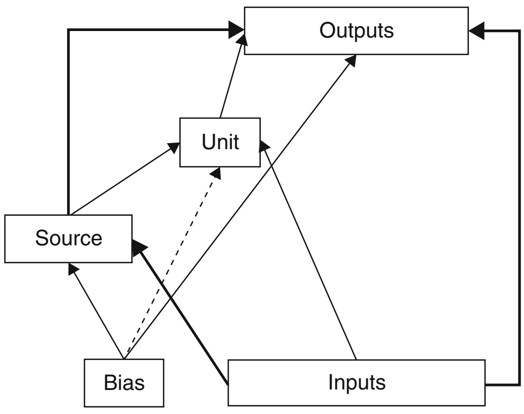
Knowledge-Based Cascade–Correlation (KBCC)

KBCC is a variant that can recruit previously-learned networks or indeed any differentiable function, in competition with single hidden units (Shultz and Rivest 2001; Shultz et al. 2007). The recruit is the candidate whose output correlates best with residual network error, just as in ordinary CC. The candidate pool usually has a number of randomly initialized sigmoid units and a number of candidate source networks, i.e., networks previously trained on other tasks. The input weights to multiple copies of the source networks are usually randomly initialized to improve optimization. Of these copies, one is typically connected with an identity matrix with off-diagonal zeros, to enable quick learning of the target task when exact knowledge is available. A hypothetical KBCC network is shown in Fig. 2.

Software

Most CC algorithms are available in a variety of formats and languages, including:

CASCOR: Lisp and C implementations of cascade–correlation
http://www.cs.cmu.edu/afs/cs/project/ai-repository/ ai/areas/neural/systems/cascor/0.html
Free Lisp and C implementations of cascade–correlation
Cascade Neural Network Simulator
http://www.cs.cmu.edu/~sef/sefSoft.htm
A public domain C program that implements cascade–correlation and recurrent cascade–correlation, plus experimental versions of cascade 2 and recurrent cascade 2
LNSC cascade–correlation simulator applet



Cascade Correlation, Fig. 2 Hypothetical knowledge-based cascade–correlation (KBCC) network that has recruited a source network and then a sigmoid unit, each installed on a separate layer. The *dashed line* represents a single connection weight, *thin solid lines* represent weight vectors, and *thick solid lines* represent weight matrices

http://www.psych.mcgill.ca/perpg/fac/shultz/cdp/lnsc_applet.htm
A Java applet allowing direct comparisons of cascade–correlation and back-propagation algorithms on some benchmark problems, also permitting entry of text-edited custom training and test patterns.
LNSC Java Code Library
http://www.lnsclab.org/
Free compiled Java versions of BP, CC, SDCC, and KBCC neural network software, along with a tutorial

Applications

CC

Partly because of its ability to grow its own networks and build new learning on top of existing knowledge, CC has been used to simulate many phenomena in cognitive development. These characteristics embody the constructivism that developmental psychologists often discussed but did not formulate precisely. Simulations are typically evaluated by how well they capture the various psychological phenomena that characterize a particular domain.

The balance-scale task involves presenting a child with a rigid beam balanced on a fulcrum with pegs spaced at equal intervals to the left and right of the fulcrum. A number of identical weights are placed on a peg on the left side and a peg on the right side, and the child is asked to predict which side will descend when the beam is released from its moorings. CC networks passed through the stages observed with children and captured the so-called torque difference effect, the tendency to do better on problems with large absolute torque differences than on problems with small torque differences (Shultz et al. 1994; Shultz and Takane 2007).

The conservation task presents a child with two quantities of objects that the child judges to be equal and then transforms one set in a way that either changes that relationship or conserves it. CC networks captured four important conservation regularities (Shultz 1998):

1. A shift from nonconservation to conservation beliefs
2. A sudden spurt in performance during acquisition
3. Emergence of correct conservation judgments for small quantities before larger quantities
4. Young children’s choice of the longer row as having more items than the shorter row

Analysis of network solutions at various points in development revealed a gradual shift from perceptual (how the sets of items look) to cognitive (whether or not the transformation changed a quantity) solutions, similar to what had been found with children.

The seriation task requires a child to order a disordered collection of sticks of different lengths. CC networks passed through the four stages seen in children (total failure, partial sort, trial-and-error sort, and systematic sort) and captured the tendency for sets with smaller differences to be more difficult to sort (Mareschal and Shultz 1999). Analysis of network solutions revealed early success at the short end of the series that was gradually extended to the longer end, as in children.

The transitivity problem typically also employs sticks of different length. Here the child is trained on all pairs of sticks that are adjacent in length and then is asked to infer the relative length of untrained pairs. Five psychological regularities were captured when CC networks were trained to compare the relative sizes of adjacent pairs (Shultz and Vogel 2004):

1. Learning short or long adjacent pairs before adjacent pairs of medium length.
2. Faster inferences with pairs farther apart in length than with pairs close together in length, an effect that diminished with age. A constraint-satisfaction network module simulated reaction times by inputting the output of a CC network and settling over time cycles into a low-energy solution that satisfied the constraints supplied by connection weights and inputs, effectively cleaning up the output of the CC network.
3. Faster inferences with pairs containing the shortest or longest stick.
4. Faster inferences when the expression used in the question (e.g., shorter) is compatible with an end stick (e.g., the shortest stick) in the compared pair than when the question term (e.g., shorter) is incompatible with an end stick (e.g., the longest stick) in the compared pair.
5. Older children learned adjacent pairs faster and made inference comparisons faster and more accurately than did young children.

The computational bases for these effects were revealed by examining the pattern of connection weights within the CC network module. The pattern of these weights formed a cubic shape, symmetrical for the two sticks being compared, in which discrimination was better at the ends of the array than in the middle and became sharper with deeper learning.

Another task calls for integration of cues for moving objects, governed by the equation velocity = distance/time. Children were presented with information on two of those quantities and asked to infer the third. Three stages involved first using the quantity that varied positively with the quantity to be inferred, second adding or subtracting the known quantities, and finally multiplying or dividing the known quantities. Already documented stages were captured and others were correctly predicted by CC networks (Buckingham and Shultz 2000).

Semantic rules for deictic personal pronouns specify that *me* refers to the person using the pronoun and *you* refers to the person who is being addressed. Although most children acquire these pronouns without notable errors, a few reverse these pronouns, persistently calling themselves *you* and the mother *me*. Such reversals in children are produced by lack of opportunity to overhear these pronouns used by other people, where the shifting reference can be observed. CC networks covered these phenomena and generated predictions for effective therapy to correct reversal errors (Oshima-Takane et al. 1999).

Discrimination shift learning tasks repeatedly present pairs of stimuli with mutually exclusive attributes on several binary dimensions, such as color, shape, and position, and a child learns to select the correct stimulus in each pair, e.g., *square*. Feedback is given and learning continues until the child reaches a success criterion, e.g., 8/10 correct. Then reward contingencies shift, usually without warning. A *reversal* shift stays within the initially relevant dimension, e.g., from *square* to *circle*. A *nonreversal* shift is to another dimension, such as from *square* to *blue*. There are related tasks that use new stimulus values in the shift phase. These are called *intradimensional* shifts if the shift remains within the initial dimension, e.g., *square* to *triangle*, or *extradimensional* if there is a change to another dimension, e.g., from *square* to *yellow*. The *optional shift* task presents only two stimulus pairs in the shift phase, making it ambiguous whether the shift is a reversal or nonreversal shift. The pattern of subsequent choices allows determination of whether the child interprets this as a reversal or a nonreversal shift.

Age differences in the large literature on these shifts indicate that older children learn a reversal shift faster than a nonreversal shift, learn an intradimensional shift faster than an extradimensional shift, make a reversal shift in the optional task, and are initially impaired on unchanged pairs during a nonreversal shift. Younger children learn reversal and nonreversal shifts equally fast, learn an intradimensional shift faster than an extradimensional shift, make a nonreversal shift in the optional task, and are unimpaired on unchanged pairs during a nonreversal shift. These findings were simulated by CC networks (Sirois and Shultz 1998), which also generated predictions that were later confirmed.

When infants repeatedly experience stimuli from a particular class, their attention decreases, but it recovers to stimuli from a different class. This familiarize-and-test paradigm is responsible for most of the discoveries of infant psychological abilities. CC networks simulated findings on infant attention to syntactic patterns in an artificial language (Shultz and Bale 2006) and age differences in infant categorization of visual stimuli (Shultz and Cohen 2004) and generated several predictions, some of which were tested and confirmed.

SDCC

Because of SDCC's ability to create a variety of network topologies, it is beginning to be used in psychology simulations: infant learning of word-stress patterns in artificial languages (Shultz and Bale 2006), syllable boundaries (Shultz and Bale 2006), visual concepts (Shultz 2006), and false-belief tasks; learning the structure of mathematical groups (Schlimm and Shultz 2009); replication of the results of the CC simulation of conservation acquisition (Shultz 2006); and concept acquisition.

CC and SDCC networks capture developmental stages by growing in computational power and by being sensitive to statistical patterns in the training environment (Shultz 2003). The importance of growth was demonstrated by comparisons with static backprop networks, designed with the same final topology as successful CC networks, that learn only by adjusting connection weights (Shultz 2006). Coupled with the variety of successful SDCC topologies, this suggests that the constructive process is more important than precise network topologies. Capturing stages is challenging because the system has to not only succeed on the task but also make the same mistakes on the road to success that children do. CC and SDCC arguably produced the best data coverage of any models applied to the foregoing phenomena. Both static and constructive networks capture various perceptual effects by virtue of their sensitivity to quantitative variation in stimulus inputs (Shultz 2003).

Comparison of the two algorithms in psychological modeling indicates that SDCC provides the same functionality as CC but with fewer connection weights and shallower and more variable network topologies (Shultz 2006).

KBCC

KBCC also has potential for simulating psychological development, but it has so far been applied mainly to toy and engineering problems. Exploration of a variety of toy problems was important in understanding the behavior of this complex algorithm. Some toy problems involved learning about two-dimensional geometric shapes under various transformations such as translation, rotation, and size changes as well as compositions of complex shapes from simpler shapes (Shultz and Rivest 2001). Networks had to learn to distinguish points within a target shape from points outside the shape. Learning time without relevant knowledge was up to 16 times longer than with relevant knowledge on these problems. There was a strong tendency to recruit relevant knowledge whenever it was available. Direct comparison revealed that KBCC learned spatial translation problems faster than Multitask Learning networks did.

Parity problems require a network to activate an output unit only when an odd number of binary inputs are activated. When parity-4 networks were included in the candidate source pool, KBCC learned parity-8 problems (with eight binary inputs) faster and with fewer recruits than did CC networks. Parity-4 networks were recruited by these KBCC target networks whenever available.

KBCC also learned complex chessboard shapes from knowledge of simpler chessboards. As with parity, networks used simpler previous knowledge to compose a solution to a more complex problem and learning was speeded accordingly.

In a more realistic vein, KBCC networks recruiting knowledge of vowels from one sort of speaker (e.g., adult females) learned to recognize vowels spoken by other sets of speakers (e.g., children and adult males) faster than did knowledge-free networks.

KBCC learned an efficient algorithm for detecting prime numbers by recruiting previously learned knowledge of divisibility (Shultz et al. 2007). This well-known detection algorithm tests the primality of an integer n by checking if n is divisible by any integers between 2 and the integer part of $\sqrt{n}$. Starting with small primes is efficient because the smaller the prime divisor, the more composites are detected in a fixed range of integers. The candidate pool contained networks that had learned whether an integer could be divided by each of a range of integers, e.g., a divide- by-2 network, a divide-by-3 network, etc., up to a divisor of 20. KBCC target networks trained on 306 randomly selected integers from 21 to 360 recruited only source networks involving prime divisors below the square root of 360, in order from small to large divisors. KBCC avoided recruiting single hidden units, source networks with composite divisors, any divisors greater than the square root of 360 even if prime, and divisor networks with randomized connection weights. KBCC never recruited a divide-by-2 source network because it instead learned to check the last binary digit of n to determine if n was odd or even, an effective shortcut to dividing by 2. Such KBCC networks learned the training patterns in about one third the time required by knowledge-free networks, with fewer recruits on fewer network layers, and they generalized almost perfectly to novel test integers. In contrast, even after mastering the training patterns, CC networks generalized less well than automatic guessing that the integer was composite, which was true for 81 % of integers in this range. As predicted by the simulation, adults testing primality also used mainly prime divisors below $\sqrt{n}$ and ordered divisors from small to large.

This work underscores the possibility of neural network approaches to compositionality because KBCC effectively composed a solution to prime number detection by recruiting and organizing previously learned parts of the problem, in the form of divisibility networks.

Future Directions

One new trend is to inject symbolic rules or functions into KBCC source networks. This is similar to KBANN, but more flexible because a KBCC target network decides whether and how to recruit these functions. This provides one method of integrating symbolic and neural computation and allows for simulation of the effects of direct instruction.

Cross-References

▶ Artificial Neural Networks
▶ Backpropagation

Recommended Reading

Baluja S, Fahlman SE (1994) Reducing network depth in the cascade-correlation learning architecture. School of Computer Science, Carnegie Mellon University, Pittsburgh

Buckingham D, Shultz TR (2000) The developmental course of distance, time, and velocity concepts: a generative connectionist model. J Cognit Dev 1:305–345

Dandurand F, Berthiaume V, Shultz TR (2007) A systematic comparison of flat and standard cascade-correlation using a student-teacher network approximation task. Connect Sci 19:223–244

Fahlman SE (1988) Faster-learning variations on back-propagation: an empirical study. In: Touretzky DS, Hinton GE, Sejnowski TJ (eds) Proceedings of the 1988 connectionist models summer school. Morgan Kaufmann, Los Altos, pp 38–51

Fahlman SE (1991) The recurrent cascade-correlation architecture. In: Touretzky DS (ed) Advances in neural information processing systems, vol 3. Morgan Kaufmann, Los Altos

Fahlman SE, Lebiere C (1990) The cascade-correlation learning architecture. In: Touretzky DS (ed) Advances in neural information processing systems, vol 2. Morgan Kaufmann, Los Altos, pp 524–532

Mareschal D, Shultz TR (1999) Development of children's seriation: a connectionist approach. Connect Sci 11:149–186

Oshima-Takane Y, Takane Y, Shultz TR (1999) The learning of first and second pronouns in En-

glish: network models and analysis. J Child Lang 26:545–575
Schlimm D, Shultz TR (2009) Learning the structure of abstract groups. In: Taatgen NA, Rijn HV (eds) Proceedings of the 31st annual conference of the cognitive science society. Cognitive Science Society, Austin, pp 2950–2955
Shultz TR (1998) A computational analysis of conservation. Dev Sci 1:103–126
Shultz TR (2003) Computational developmental psychology. MIT, Cambridge
Shultz TR (2006) Constructive learning in the modeling of psychological development. In: Munakata Y, Johnson MH (eds) Processes of change in brain and cognitive development: attention and performance XXI. Oxford University Press, Oxford, pp 61–86
Shultz TR, Bale AC (2006) Neural networks discover a near-identity relation to distinguish simple syntactic forms. Minds Mach 16:107–139
Shultz TR, Cohen LB (2004) Modeling age differences in infant category learning. Infancy 5:153–171
Shultz TR, Mareschal D, Schmidt WC (1994) Modeling cognitive development on balance scale phenomena. Mach Learn 16:57–86
Shultz TR, Rivest F (2001) Knowledge-based cascade-correlation: using knowledge to speed learning. Connect Sci 13:1–30
Shultz TR, Rivest F, Egri L, Thivierge J-P, Dandurand F (2007) Could knowledge-based neural learning be useful in developmental robotics? The case of KBCC. Int J Humanoid Robot 4:245–279
Shultz TR, Takane Y (2007) Rule following and rule use in simulations of the balance-scale task. Cognition 103:460–472
Shultz TR, Vogel A (2004) A connectionist model of the development of transitivity. In: Proceedings of the twenty-sixth annual conference of the cognitive science society. Erlbaum, Mahwah, pp 1243–1248
Sirois S, Shultz TR (1998) Neural network modeling of developmental effects in discrimination shifts. J Exp Child Psychol 71:235–274

Cascor

▶ Cascade Correlation

Case

▶ Instance

Case-Based Learning

▶ Instance-Based Learning

Case-Based Reasoning

Susan Craw
Robert Gordon University, Aberdeen, UK

Abstract

Case-based reasoning (CBR) solves problems by retrieving similar, previously solved problems and reusing their solutions. The case base contains a set of cases, and each case holds knowledge about a problem or situation, together with its corresponding solution or action. The case base acts as a memory, remembering is achieved using similarity-based retrieval, and the retrieved solutions are reused. Newly solved problems may be retained in the case base and so the memory is able to grow as problem-solving occurs.

CBR reuses remembered experiences, where the experience need not record *how* the solution was reached, simply that the solution was used for the problem. The reliance on stored experiences means that CBR is particularly relevant in domains which are ill defined, not well understood, or where no underlying theory is available. CBR systems are a useful way to capture corporate memory of human expertise.

The fundamental assumption of CBR is that *similar problems have similar solutions*: a patient with similar symptoms will have the same diagnosis, the price of a house with similar accommodation and location will be similar, the design for a kitchen with a similar shape and size can be reused, and a journey plan is similar to an earlier trip. A related assumption is that the world is a regular place, and what holds true today will probably be true tomorrow. A further assumption relevant to memory is that situations repeat, because if they do not, there is no point remembering them!

Synonyms

Experience-based reasoning; Lessons-learned systems; Memory-based learning

Theory/Solution

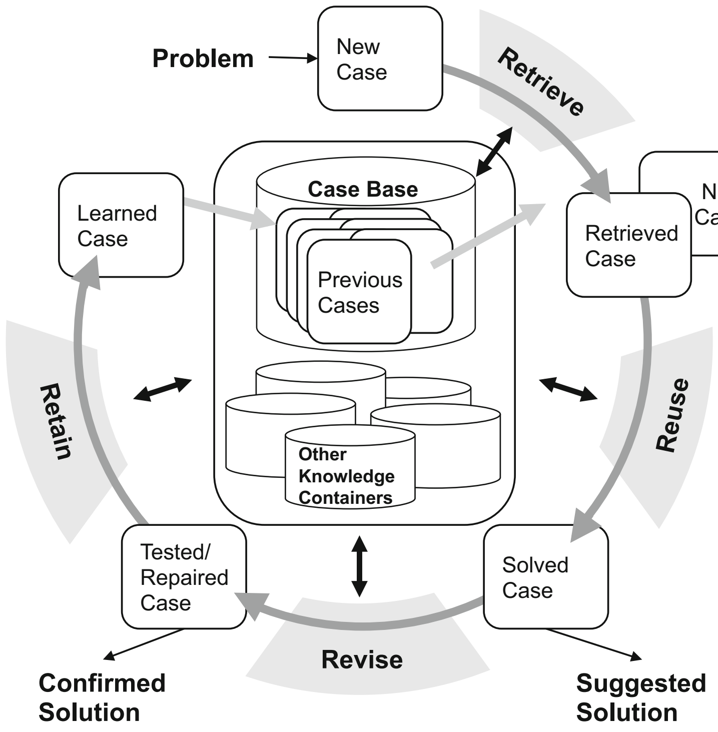
Case-based reasoning (CBR) is inspired by memory-based human problem-solving in which instances of earlier problem-solving are remembered and applied to solve new problems. For example, in case law, the decisions in trials are based on legal precedents from previous trials. In this way, specific experiences are memorized, and remembered and reused when appropriate. This contrasts with rule-based or theory-based problem-solving in which knowledge of *how* to solve a problem is applied. A doctor diagnosing a patient's symptoms may apply knowledge about how diseases manifest themselves, or she may remember a previous patient who demonstrated similar symptoms and thus apply a case-based approach.

CBR is an example of ▶ lazy learning because there is no learned model to apply to solve new problems. Instead, the generalization needed to solve unseen problems happens when a new problem is presented and the similarity-based retrieval identifies relevant previous experiences.

Figure 1 shows the CBR problem-solving cycle proposed by Aamodt and Plaza (1994). A case base of Previous Cases is the primary knowledge source in a CBR system, with additional knowledge being used to identify similar cases in the Retrieve stage, and to Reuse and Revise the retrieved case. A CBR system learns as it solves new problems when a Learned Case is created from the New Case and its Confirmed Solution, and Retained as a new case in the case base.

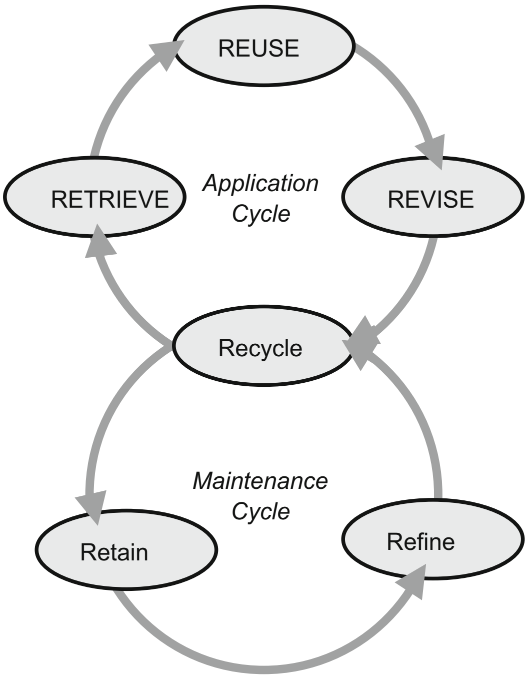
Aamodt and Plaza's four-stage CBR cycle for problem-solving and learning is commonly referred to as the "Four REs" or "R^4" cycle to recognize the following stages in Fig. 1:

– Retrieve: The cases that are most similar to the New Case defined by the description of the new problem are identified and retrieved from the case base. The Retrieve stage uses



Case-Based Reasoning, Fig. 1 CBR cycle (Adapted from Aamodt and Plaza 1994) the similarity knowledge container in addition to the case base.

- Reuse: The solutions in the Retrieved (most similar) Cases are reused to build a Suggested Solution to create the Solved Case from the New Case. The Reuse stage may use the adaptation knowledge container to refine the retrieved solutions.
- Revise: The Suggested Solution in the Solved Case is evaluated for correctness and is repaired if necessary to provide the Confirmed Solution in the Tested/Repaired Case. The Revise stage may be achieved manually or may use adaptation knowledge, but it should be noted that a revision to a Suggested Solution is likely to be a less demanding task than solving the problem from scratch.
- Retain: The Repaired Case may be retained in the case base as a newly Learned Case if it is likely to be useful for future problem-solving. Thus the primary knowledge source for CBR may be added to during problem-solving and is an evolving, self-adaptive collection of problem-solving experiences.



Case-Based Reasoning, Fig. 2 Six REs CBR cycle (Adapted from Gokër and Roth-Berghofer 1999)

This "Four REs" cycle simply Retained the Tested/Repaired Case as a new Learned Case. More recently, the Retain stage has been replaced with a Recycle-Retain-Refine loop of a "Six REs" cycle proposed by Gokër and Roth-Berghofer (1999) and shown in Fig. 2. Learned Cases are Recycled as potential new cases, the Retain step validates their correctness, before the Refine stage decides if the case should be integrated and how this should be done. The new case may be added, used to replace redundant cases, or merged with existing cases, and other case base maintenance may be required to maintain the integrity of the CBR system. The maintenance cycle is often executed separately from the problem-solving Application Cycle.

Knowledge Containers

Case knowledge is the primary source of knowledge in a CBR system. However, case knowledge is only one of four knowledge containers identified by Richter (2009):

- Vocabulary: The representation language used to describe the cases captures the concepts involved in the problem-solving.
- Similarity knowledge: The similarity measure defines how the distances between cases are computed so that the nearest neighbors are identified for retrieval.
- Adaptation knowledge: Reusing solutions from retrieved cases may require some adaptation to enable them to fit the new problem.
- Case base: The stored cases capture the previous problem-solving experiences.

The content of each knowledge container is not fixed, and knowledge in one container can compensate for lack of knowledge in another. It is easy to see that a more sophisticated knowledge representation could be less demanding on the content of the case base. Similarly, vocabulary can make similarity assessment during retrieval easier, or a more complete case base could reduce the demands on adaptation during reuse. Further knowledge containers are proposed by others (e.g., maintenance by Gokër and Roth-Berghofer 1999) .

Cases may be represented as simple feature vectors containing nominal or numeric values. A case capturing a whisky-tasting experience might contain features such as sweetness, peatiness, color, nose and palate, and the ▶ classification as the distillery where it was made.

Sweet-ness	Peati-ness	Color	Nose	Palate	Distillery
6	5	amber	full	medium dry	Dalmore

More structured representations can use frame-based or object-oriented cases. The choice of representation depends on the complexity of the experiences being remembered and is influenced by how similarity should be determined. Hierarchical case representations allow cases to be remembered at different levels of abstraction, and retrieval and reuse may occur at these different levels.

For ▶ classification tasks, the case base can be considered to contain exemplars of problem-solving. This notion of exemplar confirms a CBR case base as a source of knowledge; it contains only those experiences that are believed to be useful for problem-solving. A similar view is taken for non-classification domains where the case base contains useful prototypes: for example, designs that can be used for redesign, plans for replanning, etc.

One of the advantages of CBR is that a case base is composed of independent cases that each captures some local problem-solving knowledge that has been experienced. Therefore, the "knowledge acquisition bottleneck" of many rule-based and model-based systems is reduced for CBR. However, the other knowledge containers provide additional knowledge acquisition demands that may lessen the advantage of CBR for some domains.

Retrieval

CBR retrieval compares the problem part of the new case with each of the cases in the case base to establish the distance between the new case and the stored cases. The cases closest to the new case are retrieved for reuse. Retrieval is a major focus of López de Mántaras et al.'s (2005) review of research contributions related to the CBR cycle.

Similarity- and distance-based neighborhoods are commonly used interchangeably when discussing CBR retrieval. Similarity and distance are inverses: the similarity is highest when the distance is close to 0, and the similarity is 0 when the distance is large. Several functions may be applied to define a suitable relationship between a distance d and a similarity s, including the following simple versions:

Inverse: $s = \dfrac{1}{d+1}$

Linear: $s = 1 - d$ for normalized d $(0 \leq d \leq 1)$

It is common to establish the distance between each pair of feature values and then to use a distance metric, often Euclidean or ▶ Manhattan distance (see also ▶ Similarity Measures), to calculate the distance between the feature vectors for the New and Retrieved Cases. The distance between two numeric feature values v and w for a feature F is normally taken to be the distance between the normalized values:

$$d(v,w) = \frac{|\, v - w \,|}{F_{\max} - F_{\min}}$$

where $F_{\max}/F_{\min}$ are the maximum/minimum values of the feature F.

For nominal values v and w, the simplest approach is to apply a binary distance function:

$$d(v,w) = \begin{cases} 0 \text{ if } v = w \\ 1 \text{ otherwise} \end{cases}$$

For ordered nominal values, a more fine-grained distance may be appropriate. The distance between the ith value v_i and the jth value v_j in the ordered values $v_1, v_2, \ldots, v_n$ may use the separation in the ordering to define the distance:

$$d(v_i, v_j) = \frac{|\, i - j \,|}{n-1}.$$

Extending this to arbitrary nominal values, a distance matrix D may define the distance between each pair of nominal values by assigning the distance $d(v_i, v_j)$ to d_{ij}. Alternatively there may be background knowledge in the form of an ontology or concept hierarchy where their depth D in the structure compared to the depth of their least common ancestor (lca) can provide a measure of separation:

$$d(v_i, v_j) = \frac{D(v_i) + D(v_j)}{2D(\text{lca})}$$

Returning to the whisky-tasting example, suppose sweetness and peatiness score values 0–10, color takes ordered values {pale, straw, gold, honey, amber}, palate uses binary distance, and nose is defined by the following distance matrix:

Nose Distance Matrix

Distances	peat	fresh	soft	full
peat	0	0.3	1	0.5
fresh	0.3	0	0.5	0.7
soft	1	0.5	0	0.3
full	0.5	0.7	0.3	0

Dalmore whisky above can be compared with Laphroaig and The Macallan as follows:

Sweetness	Peatiness	Color	Nose	Palate	Distillery
2	10	amber	peat	medium dry	Laphroaig
7	4	gold	full	big body	The Macallan

The Manh distances are:

$$\begin{aligned} d(\text{Dalmore, Laphroaig}) &= 0.4 + 0.5 + 0 + 0.5 \\ &\quad + 0 = 1.4; \\ d(\text{Dalmore, The Macallan}) &= 0.1{+}0.1{+}0.5{+}0 \\ &\quad + 1 = 1.7. \end{aligned}$$

Taking all the whisky features with equal importance, Dalmore is more similar to Laphroaig than to The Macallan.

In situations where the relative importance of features should be taken into account, a weighted version of the distance function should be used; for example, the weighted Manhattan distance between two normalized vectors $\mathbf{x} = (x_1, x_2, \ldots x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots y_n)$ with weight w_i for the ith feature is

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

In the example above, if Peatiness has weight 4 and the other features have weight 1, then the weighted Manhattan distances are:

$$\begin{aligned} d(\text{Dalmore, Laphroaig}) &= (0.4 + 4 \times 0.5 + 0 \\ &\quad +0.5{+}0)/8{=}0.36; \\ d(\text{Dalmore, The Macallan}) &= (0.1{+}4{\times}0.1{+}0.5 \\ &\quad +0 + 1)/8{=}0.25. \end{aligned}$$

Therefore, emphasizing the distinctive Peatiness feature, Dalmore is more similar to The Macallan than to Laphroaig.

The similarity knowledge container contains knowledge to calculate similarities. For simple feature vectors, a weighted sum of distances is often sufficient, and the weights are similarity knowledge. However, even our whisky-tasting domain had additional similarity knowledge containing the distance matrix for the nose feature. Structured cases require methods to calculate the similarity of two cases from the similarities of components. CBR may use very knowledge-intensive methods to decide similarity for the retrieval stage. Ease of reuse or revision may even be incorporated as part of the assessment of similarity. Similarity knowledge may also define how ▶ missing values are handled: the feature may be ignored, the similarity may be maximally pessimistic, or a default or average value may be used to calculate the distance.

A CBR case base may be indexed to avoid similarity matching being applied to all the cases in the case base. One approach uses kd trees to partition the case base according to hyperplanes. ▶ Decision Tree algorithms may be used to build the kd tree by using the cases as training data, partitioning the cases according to the chosen decision nodes and storing the cases in the appropriate leaf nodes. Retrieval first traverses the decision tree to select the cases in a leaf node, and similarity matching is applied to only this partition. Case Retrieval Nets are designed to speed up retrieval by applying spreading activation to select relevant cases. In Case Retrieval Nets, the feature value nodes are linked via similarity to each other and to cases. Indexes can speed up retrieval but they also preselect cases according to some criteria that may differ from similarity.

Reuse and Revision

Reuse may be as simple as copying the solution from the Retrieved Case. If k nearest neighbors are retrieved, then a vote of the classes predicted in the retrieved cases may be used for ▶ classification, or the average of retrieved values for ▶ regression. A weighted vote or weighted average of the retrieved solutions can take account of the nearness of the retrieved cases in the calculation. For more complex solutions, such as designs or plans, the amalgamation of the solutions from the Retrieved Cases may be more knowledge intensive.

If the New Case and the Retrieved Case are different in a significant way, then it may be that the solution from the Retrieved Case should be adapted before being proposed as a Suggested Solution. Adaptation is designed to recognize significant differences between the New and Retrieved Cases and to take account of these by adapting the solution in the Retrieved Case.

In classification domains, it is likely that all classes are represented in the case base. However, different problem features may alter the classification and so adaptation may correct for a lack of cases. In constructive problem-solving like design and planning, however, it is unlikely that all solutions (designs, plans, etc.) will be represented in the case base. Therefore, a retrieved case suggests an initial design or plan, and adaptation alters it to reflect novel feature values.

There are three main types of adaptation that may be used, as part of the reuse step to refine the solution in the Retrieved Case to match better the new problem, or as part of the revise stage to repair the Suggested Solution in the Solved Case:

– Substitution: Replace parts of the retrieved solution. In Hammond's (1990) CHEF system to plan Szechuan recipes, the substitution of ingredients enables the requirements of the new menu to be achieved. For example, the beef and broccoli in a retrieved recipe are substituted with chicken and snowpeas.
– Transformation: Add, change, or remove parts of the retrieved solution. CHEF adds a skinning step to the retrieved recipe that is needed for chicken but not for beef.
– Generative Adaptation: Replay the method used to derive the retrieved solution. Thus the retrieved solution is not adapted but a new solution is generated from reusing the retrieved method for the new circumstances. This approach is similar to reasoning by analogy.

CHEF also had a clear REVISE stage where the Suggested Solution recipe was tested in simulation and any faults were identified, explained, and repaired using repair templates for different types of explained failures. In one recipe a strawberry soufflé was too liquid, and one repair is to drain the strawberry pulp, and this transformation adaptation is one REVISE operation that could be applied.

The adaptation knowledge container is an important source of knowledge for some CBR systems, particularly for design and planning, where refining an initial design or plan is expected. Acquiring adaptation knowledge can be onerous, and learning adaptation knowledge from the cases in the case base or from background knowledge of the domain has been effective (Craw et al. 2006; Jalali and Leake 2013).

Retention and Maintenance

The retention of new cases during problem-solving is an important advantage of CBR systems. However, it is not always advantageous to retain all new cases. The ▶ Utility Problem – *that the computational benefit from additional knowledge must not outweigh the cost of applying* *it* – in CBR refers to cases and the added cost of retrieval. The case base must be kept "lean and mean," and so new cases are not retained automatically, and cases that are no longer useful are removed. New cases should be retained if they add to the competence of the CBR system by providing problem-solving capability in an area of the problem space that is currently sparse. Conversely, existing cases should be reviewed for the role they play, and forgetting cases is an important maintenance task. Existing cases may contain outdated experiences and so should be removed, or they may be superseded by new cases.

Case base maintenance manages the contents of the case base to achieve high competence. Competence depends on the domain and may involve:

- quality of solution;
- user confidence in solution; or
- efficiency of solution prediction (e.g., speed-up learning).

As a result, the RETAIN step in Aamodt and Plaza's (1994) "four REs" problem-solving cycle is normally replaced by some form of case base maintenance cycle, such as the ReCycle-Retain-Refine loop in Gokër and Roth-Berghofer's (1999) "six REs" cycle.

Case base maintenance systems commonly assume that the case base contains a representative sample of the problem-solving experiences. They exploit this by using a leave-one-out approach where repeatedly for each case in the case base, the one extracted case is used as a new case to be solved, and the remaining cases become the case base. This enables the problem-solving competence of the cases in the case base to be estimated using the extracted cases as representative new cases to be solved. Various researchers build a competence model for the case base by identifying groups of cases with similar problem-solving ability and use this model to underpin maintenance algorithms that prioritize cases for deletion and to identify areas where new cases might be added.

There are several trade-offs to be managed by case base maintenance algorithms: larger case bases contain more experiences but take longer for retrieval; smaller case bases are likely to lack some key problem-solving ability; cases whose solution is markedly different from their nearest neighbors may be noisy or may be an important outlier. The competence of a case depends on other knowledge containers, and so case base maintenance should not proceed in isolation.

CBR Applications and Tools

Two notable successful deployed applications of CBR are Verdande's Drilledge that monitors oil-well drilling operations to reduce non-productive time (Gundersen et al. 2013), and General Electric's FormTool for plastic color matching (Cheetham 2005). Many more applications are described in the *Fielded Applications of CBR* article in Knowledge Engineering Review 20(3) CBR Special Issue (2005) and Montani and Jain's *Successful Case-Based Reasoning Applications* texts (Springer, 2010 & 2014):

- *Classification* – Medical diagnosis systems include SHRINK for psychiatry, CASEY for cardiac disease, and ICONS for antibiotic therapy for intensive care. Other diagnostic systems include failure prediction of rails for Dutch railways, Boeing's CASSIOPÉE for trouble-shooting aircraft engines, and the HOMER Help-Desk (Gokër and Roth-Berghofer 1999).
- *Design* – Architectural design was a popular early domain: ARCHIE and CADsyn. Other design applications include CADET and KRITIK for engineering design, pharmaceutical tablet formulation, Déjà Vu for plant control software, and Lockheed's CLAVIER for designing layouts for autoclave ovens.
- *Planning* – PRODIGY is a general purpose planner that uses analogical reasoning to adapt retrieved plans. Other planning applications include PARIS for manufacturing planning, mission planning for US navy, and route planning for DaimlerChrysler cars. A recent focus is planning in simulated complex environments as found in Real-Time Strategy Games (Jaidee et al. 2013; Ontañón and Ram

2011; Wender and Watson 2014). CBR has other Game AI applications including robot soccer and poker.

– *Textual CBR* – Legal decision support systems were an important early application domain for textual CBR, including HYPO, GREBE, and SMILE. Question answering was another fruitful text-based domain: FAQ-Finder and FA11Q. More recently, textual CBR is used for industrial decision support based on textual reports; e.g., incident management and Health & Safety.
– *Conversational CBR* – Conversational systems extract the problem specification from the user through an interactive case-based dialogue. Examples include help-desk support, CBR Strategist for fault diagnosis, and Wasabi and ShowMe product recommender systems.
– *Recommender Systems* – There has been a large growth in the use of CBR for recommendation of products, travel planning, and online music. Current topics include preference recommenders for individuals and groups (Quijano-Sánchez et al. 2012) and sentiment/opinion mining from social media to improve personalization (Dong et al. 2014).
– *Workflows* – A recent interest in process-oriented CBR has used the CAKE Collaborative Agile Knowledge Engine to create office workflows (Minor et al. 2014). Other applications include science workflows, medical pathways, modeling interaction traces, and recipes. These applications use structured cases and demand knowledge-rich adaptation for reuse. An annual Computer Cooking Competition at recent ICCBR conferences has encouraged the development of various case-based recipe systems including Taaable, JADAWeb, CookIIS, ChefFroglingo, GoetheShaker (cocktails), and EARL (sandwiches).

There are two main open-source CBR tools: myCBR and Colibri. Both provide state-of-the-art CBR functionality, and Colibri also incorporates a range of facilities for textual CBR. The myCBR tool originated from the INRECA methodology, and its website www.mycbr-project.net offers downloads, documentation, tutorials, and publications. Similar Colibri information is available at gaia.fdi.ucm.es/research/colibri, with the jColibri framework also available from www.sourceforge.net. Empolis is one of the pioneers in CBR with CBR Works being one of the first commercial CBR tools. It is now part of Empolis' Information Access System, and is available at www.empolis.com.

Future Directions

The drivers for ubiquitous computing – wireless communication and small devices – also affect future developments in CBR. The local, independent knowledge of case bases makes mobile devices ideal to collect experiences and to deliver experience-based knowledge for reuse.

Textual CBR systems are becoming increasingly important for extracting and representing knowledge captured in textual documents. This is particularly influenced by the availability of electronic documents in the Web and social media as sources of data for the extraction of representation knowledge. They also provide background knowledge from which to learn knowledge for similarity and adaptation containers.

Cross-References

▸ Instance-Based Learning
▸ Lazy Learning
▸ Nearest Neighbor
▸ Similarity Measures

Recommended Reading

Aamodt A, and Plaza E (1994) Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Commun 7:39–59. citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.1670

Cheetham W (2005) Tenth anniversary of the plastics color formulation tool. AI Mag 26(3):51–61 www.aaai.org/Papers/Magazine/Vol26/26-03/AIMag26-03-007.pdf

Craw S, Wiratunga N, Rowe RC (2006) Learning adaptation knowledge to improve case-based reasoning. Artif Intell 170(16–17):1175–1192. doi:10.1016/j.artint.2006.09.001

Dong R, Schaal M, O'Mahony MP, McCarthy K, Smyth B (2014) Further experiments in opinionated product recommendation. In: Lamontagne L, Plaza E (eds) Proceedings of the 22nd international conference on case-based reasoning, Cork. LNAI, vol 8765. Springer, Berlin/Heidelberg, pp 110–124. doi:10.1007/978-3-319-11209-1_9

Gokër MH, Roth-Berghofer T (1999) The development and utilization of the case-based help-desk support system HOMER. Eng Appl Artif Intell 12:665–680. doi:10.1016/S0952-1976(99)00037-8

Gundersen OE, Sørmo F, Aamodt A, Skalle P (2013) A real-time decision support system for high cost oil-well drilling operations. AAAI AI Mag 34(1): 21–31. www.aaai.org/ojs/index.php/aimagazine/article/view/2434

Hammond KJ (1990) Explaining and repairing plans that fail. Artif Intell 45(1–2):173–228

Jaidee U, Muñoz-Avila H, Aha DW (2013) Case-based goal-driven coordination of multiple learning agents. In: Delaney SJ, Ontanon S (eds) Proceedings of the 21st international conference on case-based reasoning, Saratoga Springs. LNAI, vol 7969. Springer, Berlin/Heidelberg, pp 164–178. doi:10.1007/978-3-642-39056-2_12

Jalali V, Leake D (2013) Extending case adaptation with automatically-generated ensembles of adaptation rules. In: Delaney SJ, Ontanon S (eds) Proceedings of the 21st international conference on case-based reasoning, Saratoga Springs. LNAI, vol 7969. Springer, Berlin/Heidelberg, pp 188–202. doi:10.1007/978-3-642-39056-2_14

López de Mántaras R, McSherry D, Bridge D, Leake D, Smyth B, Craw S, Faltings B, Maher ML, Cox MT, Forbus K, Aamodt A, Watson I (2005) Retrieval, reuse, revision, and retention in case-based reasoning. Knowl Eng Rev 20(3):215–240. doi:10.1017/S0269888906000646

Minor M, Bergmann R, Görg S (2014) Case-based adaptation of workflows. Inf Syst 40:142–152. doi:10.1016/j.is.2012.11.011

Ontañón S, Ram A (2011) Case-based reasoning and user-generated AI for real-time strategy games. In: Artificial intelligence for computer games. Springer, New York, pp 103–124. doi:10.1007/978-1-4419-8188-2_5

Quijano-Sánchez L, Bridge D, Díaz-Agudo B, Recio-García JA (2012) Case-based aggregation of preferences for group recommenders. In: Díaz-Agudo B, Watson I (eds) Proceedings of the 20th international conference on case-based reasoning, Lyon. LNAI, vol 7466. Springer, Berlin/Heidelberg, pp 17–31. doi:10.1007/978-3-642-32986-9_25

Richter MM (2009) The search for knowledge, contexts, and case-based reasoning. Eng Appl Artif Intell 22(1):3–9. doi:10.1016/j.engappai.2008.04.021

Wender S, Watson I (2014) Combining case-based reasoning and reinforcement learning for unit navigation in real-time strategy game AI. In: Lamontagne L, Plaza E (eds) Proceedings of the 22nd international conference on case-based reasoning, Cork. LNAI, vol 8765. Springer, Berlin/Heidelberg, pp 511–525. doi:10.1007/978-3-319-11209-1_36

Categorical Attribute

Synonyms

Qualitative attribute

Categorical attributes are attributes whose values can be placed into distinct categories See ▸ Attribute and ▸ Measurement Scales.

Categorical Data Clustering

Periklis Andritsos[1] and Panayiotis Tsaparas[2]
[1]Faculty of Information, University of Toronto, Toronto, ON, Canada
[2]Department of Computer Science & Engineering, University of Ioannina, Ioannina, Greece

Abstract
In this chapter, we provide an overview of the categorical data clustering problem. We first present different techniques for the general cluster analysis problem, and then study how these techniques specialize to the case of non-numerical (categorical) data. We also present measures and techniques developed specifically for this domain.

Synonyms

Clustering of nonnumerical data; Grouping

Definition

Data clustering is informally defined as the problem of partitioning a set of objects into groups, such that objects in the same group are similar, while objects in different groups are dissimilar. Categorical data clustering refers to the case where the data objects are defined over *categorical* attributes. A categorical attribute is an attribute whose domain is a set of discrete values that are not inherently comparable. That is, there is no single ordering or inherent distance function for the categorical values, and there is no mapping from categorical to numerical values that is semantically sensible.

Motivation and Background

Clustering is a problem of great practical importance that has been the focus of substantial research in several domains for decades. As the volume of collected data grows, the need to mine and understand the data becomes imperative. Clustering plays an instrumental role in this process. As a result in the recent years, there has been a surge of research activity in devising new clustering algorithms that can handle large amounts of data and produce results of high quality.

In data clustering we want to partition objects into groups such that similar objects are grouped together, while dissimilar objects are grouped separately. This definition assumes that there is some well-defined notion of similarity, or distance, between data objects, and a way to decide if a group of objects is a homogeneous cluster. Most of the clustering algorithms in the literature focus on data sets where objects are defined over numerical attributes. In such cases, the similarity (or dissimilarity) of objects can be defined using well-studied measures that are derived from geometric analogies. These definitions rely on the semantics of the data values themselves (e.g., the values $100,000 and $110,000 are more similar than $100,000 and $1). The existence of a distance measure enables us to define a quality measure for a clustering (e.g., the mean square distance between each point and the representative of its cluster). Clustering then becomes the problem of grouping together points such that the quality measure is optimized.

Categorical Data Clustering, Table 1 An instance of a movie database

	Director	Actor	Genre
$\mathbf{t}_1$ (Godfather II)	Scorsese	De Niro	Crime
$\mathbf{t}_2$ (Good fellas)	Coppola	De Niro	Crime
$\mathbf{t}_3$ (Vertigo)	Hitchcock	Stewart	Thriller
$\mathbf{t}_4$ (N by NW)	Hitchcock	Grant	Thriller
$\mathbf{t}_5$ (Bishop's wife)	Koster	Grant	Comedy
$\mathbf{t}_6$ (Harvey)	Koster	Stewart	Comedy

However, there are many data sets where the data objects are defined over attributes, which are neither numerical nor inherently comparable in any way. We term such data sets *categorical*, since they represent values of certain categories. As a concrete example, consider the toy data set in Table 1 that stores information about movies. For the purpose of exposition, a movie is characterized by the attributes "director," "actor/actress," and "genre." In this setting, it is not immediately obvious what the distance, or similarity, is between the values "Coppola" and "Scorsese" or the tuples "Vertigo" and "Harvey."

There are plenty of examples of categorical data: product data, where products are defined over attributes such as brand, model, or color; census data, where information about individuals includes attributes such as marital status, address, and occupation; ecological data where plants and animals can be described with attributes such as shape of petals or type of habitat. There is a plethora of such data sets, and there is always a need for clustering and analyzing them.

The lack of an inherent distance or similarity measure between categorical data objects makes clustering of categorical data a challenging problem. The challenge lies in defining a quality measure for categorical data clustering that captures the human intuition of what it means for categorical data objects to be similar. In the following, we will present an overview of the different efforts at addressing this problem and the resulting clustering algorithms.

Structure of the Learning System

Generic Data Clustering System

We first describe the outline for a generic data clustering system, not necessarily of categorical data. We will then focus on techniques specific to categorical data clustering.

Data clustering is not a one-step process. In one of the seminal texts on cluster analysis, Jain and Dubes (1988) divided the clustering process into seven different stages starting from the data collection and ending with the result interpretation. In this article we are interested in problems relating to the representation of the data, which affects the notion of similarity between the categorical tuples and the clustering strategy, which determines the final algorithm. These lie in the heart of the data clustering problem, and there has been considerable research effort in these areas within the data mining and machine learning communities. More specifically we will consider the following two subproblems: (a) the formal formulation of the clustering problem and (b) the clustering algorithm.

Formal formulation of the clustering problem: In order to devise algorithms for clustering, we need to mathematically formulate the intuition behind the informal definition of the clustering problem where "similar objects are grouped together and dissimilar objects are grouped separately." The problem formulation typically requires at least one of the following:

- A measure of similarity or distance between two data objects.
- A measure of similarity or distance between a data object and a cluster of objects. This is often defined by defining a representative for a cluster as a (new) data object and comparing the data object with the representative.
- A measure of the quality of a cluster of data objects.

The result of the formulation step is to define a clustering optimization criterion that guides the grouping of the objects into clusters.

When the data is defined over numerical attributes, these measures are defined using geometric analogies. For example, in one possible formulation, we can view each object as a point in the Euclidean space, and define the distance between two points as the Euclidean distance, and the representative of a cluster as the mean Euclidean vector. The quality of a cluster can be defined with respect to the "variance" of the cluster, that is, the sum of squares of the distances between each object and the mean of the cluster. The optimization problem then becomes to minimize the variance over all clusters.

The clustering algorithm: Once we have a mathematical formulation of the clustering problem, we need an algorithm that will find the optimal clustering in an efficient manner. In most cases finding the optimal solution is an NP-hard problem, so there are a variety of efficient heuristics or approximation algorithms. There is an extensive literature on this subject that approaches the problem from different angles and a wide variety of different clustering techniques and algorithms. We now selectively describe some broad classes of clustering algorithms and problems. A thorough categorization of clustering techniques can be found in Han and Kamber (2001), where different clustering problems, paradigms, and techniques are discussed.

Hierarchical clustering algorithms: This is a popular clustering technique since it is easy to implement, and it lends itself well to visualization and interpretation. Hierarchical algorithms create a hierarchical decomposition of the objects. They are either *agglomerative* (*bottom-up*) or *divisive* (*top-down*). *Agglomerative* algorithms start with each object being a separate cluster itself and successively merge groups according to some criterion. *Divisive* algorithms follow the opposite strategy. They start with one cluster consisting of all objects and successively split clusters into smaller ones, until each object falls in one cluster, or the desired given condition is met. The hierarchical *dendrogram* produced is often in itself the output of the algorithm, since it can be used for visualizing the data. Most of the times, both approaches suffer from the fact that once a merge or a split is committed, it cannot be undone or refined.

Partitional clustering algorithms: Partitional clustering algorithms define a clustering optimization criterion and then seek the partition that optimizes this criterion. Exhaustive search over all partitions is infeasible since even for few data objects, the number of possible partitions is prohibitively large. Partitional clustering algorithms often start with an initial, usually random, partition and proceed with its refinement by locally improving the optimization criterion. The majority of them could be considered as greedy-like algorithms. They suffer from the fact that they can easily get stuck to local optima.

Spectral clustering algorithms: Spectral algorithms use the data set to be clustered to construct a two-dimensional matrix of data objects and attributes. The entries in the matrix may be the raw values or some transformation of these values. The principal eigenvectors of the matrix are then used to reveal the clustering structure in the data. There is a rich literature on different types of spectral algorithms.

Graph clustering: Graph clustering defines a range of clustering problems, where the distinctive characteristic is that the input data is represented as a graph. The nodes of the graph are the data objects, and the (possibly weighted) edges capture the similarity or distance between the data objects. The data may come naturally in the form of a graph (e.g., a social network), or the graph may be derived in some way from the data (e.g., link two products if they appear together in a transaction). Some of the techniques we describe above are directly applicable to graph data.

Categorical Data Clustering System

In the clustering paradigm we outlined, a step of fundamental importance is to formally formulate the clustering problem, by defining a clustering optimization criterion. As we detail above, for this step we need a measure of distance or similarity between the data objects or a measure of cluster quality for a group of data objects. For categorical data there exists no inherent ordering or distance measure, and no natural geometric analogies we can explore, causing the clustering paradigm to break down. Research efforts on categorical data clustering have focused on addressing this problem by imposing distance measures on the categorical data and defining clustering quality criteria. We now outline some of these approaches.

Overlap-based similarity measures: A simple and intuitive method for comparing two categorical data objects is to view them as sets of attribute values and count the overlap between the categorical values of the objects. The higher the overlap, the more similar the two objects are. This intuition leads to the use of well-known measures such as the (*generalized*) *Hamming distance* (Jain and Dubes 1988), which measures the number of common values between two tuples, or the *Jaccard* similarity measure, which is defined as the intersection over the union of the values in the two tuples. In the example of Table 1, tuples $\mathbf{t}_1$ and $\mathbf{t}_2$ have Hamming distance 1 and Jaccard coefficient $1/2$.

Two algorithms that make use of overlap-based measures are *k-modes* (Huang 1998) and *ROCK (RObust Clustering using linKs)* (Guha et al. 1999). The k-modes algorithm is a partitional algorithm inspired by the *k-means* algorithm, a well-known clustering algorithm for numerical data. The representative of a categorical data cluster is defined to be a data object where each attribute takes the *mode* value: the mode of an attribute is the most frequent attribute value in the cluster. The ROCK algorithm makes use of the Jaccard coefficient to define *links* between data objects. The data is then represented in the form of a graph, and the problem becomes essentially a graph clustering problem. Given two clusters of categorical data, ROCK measures the similarity of two clusters by comparing their *aggregate interconnectivity* against a user-specified model, thus avoiding the problem of defining a cluster representative.

Context-based similarity measures: One way to define relationships between categorical values is by comparing the *context* in which they appear. For two categorical attribute values, we define the context as the values of other attributes with which they co-occur in the data set. The more similar these two contexts are, the more similar the attribute values are. For example, in Table 1, Scorsese and Coppola are close since they appear in exactly the same context, while Scorsese and Hitchcock are far since their contexts are completely disjoint. Defining a distance between value contexts can be done using overlap similarity measures (Das and Mannila 2000) or by using information-theoretic measures, i.e., comparing the distributions defined by the two contexts (Andritsos et al. 2004). Once we have the relationships between the values, we can use standard clustering techniques for solving the clustering problem.

There are various algorithms that make use of the idea that similar values should appear in similar contexts in order to cluster categorical data. The *CACTUS (clustering categorical data using summaries)* algorithm (Ganti et al. 1999) creates groups of attribute values based on the similarity of their context. It then uses a hierarchical greedy algorithm for grouping tuples and attributes. In a slightly different fashion, *STIRR (sieving through iterated relational reinforcement)* (Ganti et al. 1998) uses the idea that similar tuples should contain co-occurring values, and similar values should appear in tuples with high overlap. This idea is implemented via a dynamical system, inspired by information retrieval techniques. When the dynamical system is linear, the algorithm is similar to spectral clustering algorithms. *CLICKS* (Zaki et al. 2005) is an algorithm that is similar to STIRR. Rather than a measure of similarity/distance, it uses a graph-theoretic approach to find k disjoint sets of vertices in a graph constructed for a particular data set. One special characteristic of this algorithm is that it discovers clusters in a subset of the underlying set of attributes.

Information-theoretic clustering criteria: The information content in a data set can be quantified through the well-studied notions of *entropy* and *mutual information* (Cover and Thomas 1991). Entropy measures the uncertainty in predicting the values of the data when drawn from a distribution. If we view each tuple, or cluster of tuples, as a distribution over the categorical values, then we can define the *conditional entropy* of the attribute values given a set of tuples, as the uncertainty of predicting the values in this set of tuples. If we have a single tuple, then the entropy is zero, since we can accurately predict the values. For tuple $\mathbf{t}_1$ we know the director, the actor, and the genre with full certainty. As we group tuples together, the uncertainty (and entropy) increases. Grouping together tuples $\mathbf{t}_1$ and $\mathbf{t}_2$ creates uncertainty about the director attribute, while grouping $\mathbf{t}_1$ and $\mathbf{t}_3$ creates uncertainty about all attributes. Hence the latter grouping has higher entropy than the former. Information-theoretic criteria for clustering aim at generating clusters with low entropy, since this would imply that the clusters are homogeneous, and there is little *information loss*. This formulation allows for defining the distance between sets of tuples, using entropy-based divergences, such as the *Jensen-Shannon* divergence (Cover and Thomas 1991). Jensen-Shannon divergence captures the information contained in the data set, in a similar way that mean-squared distance captures geometric notions inherent in numerical data.

Two algorithms that make use of this idea are *COOLCAT* (Barbarà et al. 2002) and *LIMBO (scalable information bottleneck)* (Andritsos et al. 2004). COOLCAT is a partitional algorithm that performs a local search for finding the partition with k clusters with the lowest entropy. LIMBO works by constructing a summary of the data set that preserves as much information about the data as possible and then produce a hierarchical clustering of the summary. It is a scalable algorithm that can be used in both static and streaming environments.

A related approach is adopted by the COBWEB algorithm (Fisher 1987; Gluck and Corter 1985), a divisive hierarchical algorithm that optimizes the *category utility* measure, which measures how well particular values can be predicted given the clustering as opposed to having them in the original data set unclustered.

Categorical clustering as clustering aggregation: A different approach to the categorical data clustering problem is to view it as a *clustering aggregation problem*. Given a collection of clusterings of the data objects, the clustering aggregation problem looks for the single clustering that agrees as much as possible from the input clusterings. The problem of clustering aggregation has been shown to be equivalent to categorical data clustering (Gionis et al. 2007), where each categorical attribute defines a clustering of the data objects, grouping all the objects with the same value together. For example, in Table 1, the attribute "genre" defines three clusters: the crime cluster, the thriller cluster, and the comedy cluster. Similarly, the attribute "actor" defines three clusters, and the attribute "director" defines four clusters.

Various definitions have been considered in the literature for the notion of agreement between the output clustering and the input clusterings. One definition looks at all pairs of objects and defines a *disagreement* between two clusterings if one clustering places the two objects in the same cluster, while the other places them in different clusters; an *agreement* is defined otherwise. The clustering criterion is then to minimize the number of disagreements (or maximize the number of agreements). Other definitions are also possible, which make use of information-theoretic measures or mappings between the clusters of the two clusterings. There is a variety of algorithms for finding the best aggregate cluster, many of which have also been studied theoretically.

Cross-References

- ▶ Data mining on Text
- ▶ Instance-Based Learning

Recommended Reading

Andritsos P, Tsaparas P, Miller RJ, Sevcik KC (2004) LIMBO: scalable clustering of categorical data. In: Proceedings of the 9th international conference on extending database technology (EDBT), Heraklion, 14–18 Mar 2004, pp 123–146

Barbarà D, Couto J, Li Y (2002) COOLCAT: an entropy-based algorithm for categorical clustering. In: Proceedings of the 11th international conference on information and knowledge management (CIKM), McLean, 4–9 Nov 2002, pp 582–589

Cover TM, Thomas JA (1991) Elements of information theory. Wiley, New York

Das G, Mannila H (2000) Context-based similarity measures for categorical databases. In: Proceedings of the 4th European conference on principles of data mining and knowledge discovery (PKDD), Lyon, 13–16 Sept 2000, pp 201–210

Fisher DH (1987) Knowledge acquisition via incremental conceptual clustering. Mach Learn 2: 139–172

Ganti V, Gehrke J, Ramakrishnan R (1999) CACTUS: clustering categorical data using summaries. In: Proceedings of the 5th international conference on knowledge discovery and data mining, (KDD), San Diego, 15–18 Aug 1999, pp 73–83

Gionis A, Mannila H, Tsaparas P (2007) Clustering aggregation. In: ACM transactions on knowledge discovery from data (TKDD), Mar 2007, vol 1, No 1. Association for Computing Machinery, New York

Gibson D, Kleinberg JM, Raghavan P (1998) Clustering categorical data: an approach based on dynamical systems. In: Proceedings of the 24rth international conference on very large data bases, (VLDB), New York, 24–27 Aug 1998, pp 311–322

Gluck M, Corter J (1985) Information, uncertainty, and the utility of categories. In: Proceedings of the 7th annual conference of the Cognitive Science Society (COGSCI), Irvine, pp 283–287

Guha S, Rastogi R, Shim K (1999) ROCK: a robust clustering algorithm for categorical atributes. In: Proceedings of the 15th international conference on data engineering, Sydney, 23–26 Mar 1999, pp 512–521

Jain AK, Dubes RC (1988) Algorithms for clustering data. Prentice-Hall, Englewood Cliffs

Jarke M, Lenzerini M, Vassiliou Y, Vassiliadis P (1999) Fundamentals of data warehouses. Springer-Verlag, Berlin/Heidelberg

Han J, Kamber M (2001) Data mining: concepts and techniques. Morgan Kaufmann, San Francisco

Huang Z (1998) Extensions to the k-means algorithm for clustering large data sets with categorical values. Data Min Knowl Discov 2(3):283–304

Zaki MJ, Peters M, Assent I, Seidl T (2005) CLICKS: an effective algorithm for mining subspace clusters in categorical datasets. In: Proceeding of the 11th international conference on knowledge discovery and data mining (KDD), Chicago, 21–24 Aug 2005, pp 736–742

Categorization

▶ Classification
▶ Concept Learning

Category

▶ Class

Causal Discovery

▶ Learning Graphical Models

Causality

Ricardo Silva
Centre for Computational Statistics and Machine Learning, University College London, London, UK

Abstract
Causality is an essential concept in our understanding of the world as, in order to predict how a system behaves under an intervention, it is necessary to have causal knowledge of the impact of interventions. This knowledge should be expressed in a language built on top of probabilistic models, since the axioms of probability do not provide a way of expressing how external interventions affect a system. Learning this knowledge from data also poses additional challenges compared to the standard machine learning problem, as much data comes from passive observations that do not follow the same regime under which our predictions will take place.

Definition

The main task in causal inference is the prediction of the outcome of an intervention. For example, a treatment assigned by a doctor that will change the patient's heart condition is an intervention. Predicting the change in patient condition is a causal inference task. In general, an intervention is an action taken by an external agent that changes the original values, or the probability distributions, of some of the variables in the system. Besides predicting outcomes of actions, causal inference is also concerned with explanation: identifying which were the causes of a particular event that happened in the past.

Motivation and Background

Many problems in machine learning are prediction problems. Given a feature vector $\mathbf{X}$, the task is to provide an estimate of some output vector $\mathbf{Y}$ or its conditional probability distribution $P(\mathbf{Y}|\mathbf{X})$. This typically assumes that the distribution of $\mathbf{Y}$ given $\mathbf{X}$ during learning is the same distribution at prediction time. There are many scenarios where this is not the case.

Epidemiology and several medical sciences provide counterexamples. Consider two seemingly straightforward learning problems. In the first example, one is given epidemiological data where smokers are clearly more inclined than nonsmokers to develop lung cancer. Can I use this data to learn that smoking causes cancer? In the second example, consider a group of patients suffering from a type of artery disease. In this group, those that receive a bypass surgery are likely to survive longer than those that receive a particular set of drugs with no surgery.

There is no fundamental problem on using such datasets to predict the probability of a smoker developing lung cancer or the life expectancy of someone who went through surgery. Yet, the data does not necessarily tell you if smoking is a cause of lung cancer or that nationwide the government should promote surgery as the treatment of choice for that particular heart disease. What is going on?

There are reasons to be initially suspicious of such claims. This is well known in statistics as the expression "association is not causation" (Wasserman 2004, p. 253). The data generating mechanism for our outcome **Y** ("developing lung cancer," "getting cured from artery disease") given the relevant inputs **X** ("smoking habit," "having a surgery") might change under an *intervention* for reasons such as follows.

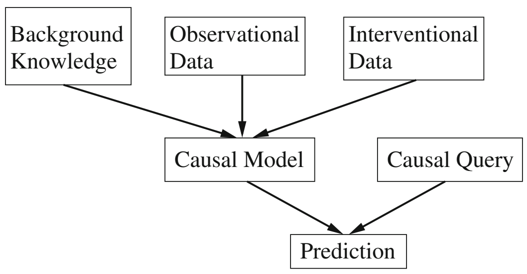
In the smoking example, the reality might be that there are several *hidden common causes* that are responsible for the observed association. A genetic factor includes, for instance, the possibility that there is a class of genotypes on which people are more likely to pick up smoking *and* develop lung cancer, without any direct causal connection between these two variables. In the artery disease example, surgery might not be the best choice to be made by a doctor. It might have been the case that so far patients in better shape were more daring in choosing, by themselves, the surgery treatment. This *selection bias* will favor surgery over drug treatment, since from the outset the patients that are most likely to improve take that treatment.

When treatment is enforced by an *external agent* (the doctor), such selection bias disappears, and the resulting $P(\mathbf{Y}|\mathbf{X})$ will not be the same. One way of learning this relationship is through *randomized trials* (Rosenbaum 2002). The simplest case consists on flipping a coin for each patient on the training set. Each face of the coin corresponds to a possible treatment, and assignment is done accordingly. Since assignment does not depend on any hidden common cause or selection bias, this provides a basis for learning causal effects. Machine learning and statistical techniques can be applied directly in this case (e.g., logistic regression). Data analysis performed with a randomized trial is sometimes called an *interventional study*.

The smoking case is more complicated: a direct intervention is not possible, since it is not acceptable to force someone to smoke or not to smoke. The inquiry asks only for a *hypothetical intervention*, i.e., if someone is forced to smoke, will his or her chances of developing lung cancer increase? Such an intervention will not take place, but this still has obvious implications in public policy. This is the heart of the matter in issues such as deciding on raising tobacco taxes or forbidding smoking in public places. However, data that measures this interventional data generation mechanism will never be available for ethical reasons. The question has to be addressed through an *observational study*, i.e., a study for causal predictions without interventional data.

Observational studies arise not only under the impossibility of performing interventions but also in the case where performing interventions is too expensive or time consuming. In this case, observational studies, or a combination of observational and interventional studies, can provide extra information to guide an experimental analysis (Hyttinen et al. 2013; Sachs et al. 2005; Cooper and Yoo 1999; Eaton and Murphy 2007). The use of observational data, or the combination of several interventional datasets, is where the greatest contributions of machine learning to causal inference rest.

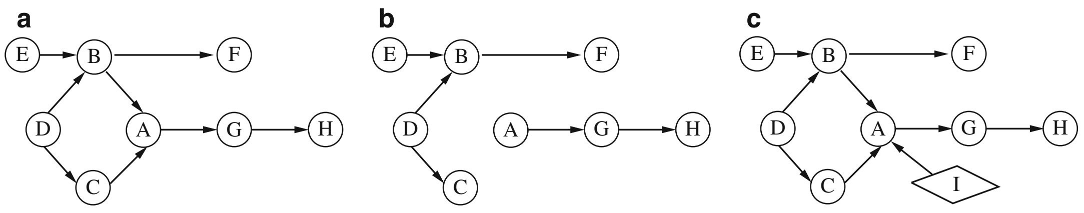
Structure of the Learning System



Structure of Causal Inference

In order to use observational data, a causal inference system needs a way of linking the state of the world under an intervention to the *natural* state of the world. The natural state is defined as the one to which no external intervention is applied. In the most general formulation, this link between the natural state and the manipulated world is defined for interventions in any subset of variables in the system.

A common language for expressing the relationship between the different states of the world is a *causal graph*, as explained in more detail in the next section. A causal model is composed of the graph and a probability distribution that



Causality, Fig. 1 (**a**) A causal DAG. (**b**) Structure of the causal graph under some intervention that sets the value of A to a constant. (**c**) Structure of the causal graph under some intervention that changes the distribution of A

factorizes according to the graph, as in a standard graphical model. The only difference between a standard graphical model and a causal graphical model is that in the latter, extra assumptions are made. The graphical model can be seen as a way of encoding such assumptions.

The combination of assumptions and observational and interventional data generates such a graphical causal model. In the related problem of reinforcement learning, the agent has to maximize a specific utility function and typically has full control on which interventions (actions) can be performed. Here we will focus on the unsupervised problem of learning a causal model for a fixed input of observational and interventional data.

Because only some (or no) interventional data might be available, the learning system might not be able to answer some causal queries. That is, the system will not provide an answer for some prediction tasks.

Languages and Assumptions for Causal Inference

Directed acyclic graphs (DAGs) are a popular language in machine learning to encode qualitative statements about causal relationships. A DAG is composed of a set of vertices and a set of directed edges. The notions of parents, children, ancestors, and descendants are the usual ones found in graphical modeling literature.

In terms of causal statements, a directed edge $A \rightarrow B$ states that A is a *direct* cause of B: that is, different interventions on A will result on different distributions for B, even if we intervene on all other variables. The assumption that A is a cause of B is not used in non-causal graphical models.

A causal DAG G satisfies the *causal Markov condition* if and only if a vertex is independent of all of its non-descendants given its direct causes (parents). In Fig. 1a, A is independent of D, E, and F given its parents, B and C. It may or may not be independent of G given B and C.

The causal Markov condition implies several other conditional independence statements. For instance, in Fig. 1a, we have that H is independent of F given A. Yet, this is not a statement about the parents of any vertex. Pearl's d-separation criterion (Pearl 2000) is a sound and complete way of reading off independencies, out of a DAG, which are entailed by the causal Markov condition. We assume that the joint probability distribution over the vertex variables is *Markov* with respect to the graph, that is, any independence statement that is encoded by the graph should imply the corresponding independence in the distribution.

Representing Interventions

The local modularity given by the causal Markov condition leads to a natural notion of intervention. Random variable V, represented by a particular vertex in the graph, is following a *local mechanism*: its direct causes determine the distribution of V before its direct effects are generated. The role of an intervention is to *override* the natural local mechanism. An external agent substitutes the natural $P(V|Parents(V))$ by a new distribution $P_{Man}(V|Parents(V))$ while keeping the rest of the model unchanged ("Man" here stands for a particular manipulation). The notion of intervening by changing a single local mechanism is sometimes known as an *ideal intervention*. Other general types of interventions can be defined (Eaton and Murphy 2007), but the most common frameworks for calculating causal effects rely on this notion.

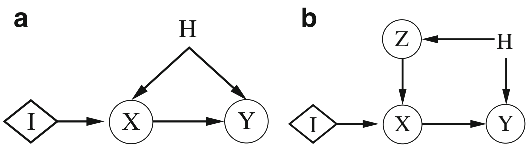
A common type of intervention is the point mass intervention, which happens when V is set to some constant v. This can be represented graphically by "wiping out" all edges into V. Figure 1b represents the resulting graph in (a) under a point manipulation of A. Notice that A is now d-separated from its direct causes under this regime. It is also probabilistically independent, since A is now constant. This allows for a graphical machinery that can read off independencies out of a *manipulated* graph (i.e., the one with removed edges). It is the idea of representing the natural state of the world with a single causal graph, and allowing for modifications in this graph according to the intervention of choice, that links the different regimes obtained under different interventions.

For the general case where a particular variable V is set to a new distribution, a *manipulation node* is added as an extra parent of V: this represents that an external agent is acting over that particular variable (Spirtes et al. 2000; Pearl 2000; Dawid 2003), as illustrated in Fig. 1c. $P(V|Parents(V))$ under intervention I is some new distribution $P_{Man}(V|Parents(V), I)$.

Calculating Distributions Under Interventions

The notion of independence is a key aspect of probabilistic graphical models, where it allows for efficient computation of marginal probabilities. In causal graphical models, it also fulfills another important role: independencies indicate that the effect of some interventions can be estimated using observational data.

We will illustrate this concept with a simple example. One of the key difficulties in calculating a causal effect is *unmeasured confounding*, i.e., hidden common causes. Consider Fig. 2a, where X is a direct cause of Y and H is a hidden common cause of both. I is an intervention vertex. Without extra assumptions, there is no way of estimating the effect of X on Y using a training set that is sampled from the observed marginal $P(X, Y)$. This is more easily seen in



Causality, Fig. 2 (**a**) X and Y have a hidden common cause H. (**b**) Y is dependent on the intervention node I given X, but conditioning on Z and marginalizing it out will allow us to eliminate the "backdoor" path that links X and Y through the hidden common cause H

the case where the model is multivariate Gaussian with zero mean. In this case, each variable is a linear combination of its parents with standard Gaussian additive noise

$$\begin{aligned} X &= aH + \epsilon_X \\ Y &= bX + cH + \epsilon_Y \end{aligned}$$

where H is also a standard normal random variable. The manipulated distribution $P_{Man}(Y|X, I)$, where I is a point intervention setting $X = x$, is a Gaussian distribution with mean $b \cdot x$. Value x is given by construction, but one needs to learn the unknown value b.

One can verify that the covariance of X and Y in the natural state is given by $a + bc$. Observational data, i.e., data sampled from $P(X, Y)$, can be used to estimate the covariance of X and Y in the natural state, but from that it is not possible to infer the value of b: there are too many degrees of freedom.

However, there *are* several cases where the probability of **Y** given some intervention on **X** can be estimated with observational data and a given causal graph. Consider the graph in Fig. 2b. The problem again is to learn the distribution of Y given X under regime I, i.e., $P(Y|X, I)$. It can be read off from the graph that I and Y are not independent given X, which means $P(Y|X, I) \neq P(Y|X)$. How can someone then estimate $P(Y|X, I)$ if no data for this process has been collected? The answer lies on *reducing the "causal query" to a "probabilistic query"* where the dependence on I disappears (and, hence, the necessity of having data measured under the I intervention). This is done by relying on the assumptions encoded by the graph:

$$\begin{aligned} P(Y|X,I) &= \textstyle\sum_z P(Y|X,I,z)P(Z=z|X,I) && \text{(Z is discrete in this example)} \\ &= \textstyle\sum_z P(Y|X,z)P(Z=z|X,I) && \text{(Y and I are independent given Z)} \\ &\propto \textstyle\sum_z P(Y|X,z)P(X|z,I)P(Z=z|I) && \text{(By Bayes' rule)} \\ &= \textstyle\sum_z P(Y|X,z)P(X|z,I)P(Z=z) && \text{(Z and I are marginally independent)} \end{aligned} \tag{1}$$

In the last line, we have $P(Y|X,Z)$ and $P(Z)$, which can be estimated with observational data, since no intervention variable I appears in the expression. $P(X|Z,I)$ is set by the external agent: its value is known by construction. This means that the causal distribution $P(Y|X,I)$ can be learned even in this case where X and Y share a hidden common cause H.

There are several notations for denoting an interventional distribution such as $P(Y|X,I)$. One of the earliest was due to Spirtes et al. (2000), which used the notation $P(Y|set\ X = x)$ to represent the distribution under an intervention I that fixed the value of X to some constant x. Pearl (2000) defines the operator *do* with an analogous purpose:

$$P(Y|do(X=x)) \tag{2}$$

Pearl's *do*-calculus is essentially a set of operations for reducing a probability distribution that is a function of some intervention to a probability distribution that does not refer to any intervention. All reductions are conditioned on the independencies encoded in a given causal graph. This is in the same spirit of the example presented above.

The advantage of such notations is that, for point interventions, they lead to simple yet effective transformations (or to a report that no transformation is possible). Spirtes et al. (2000) and Pearl (2000) provide a detailed account of such prediction tools. By making a clear distinction between $P(Y|X)$ (X under the natural state) and $P(Y|do(X))$ (X under some intervention), much of the confusion that conflates causal and noncausal predictions disappears.

It is important to stress that methods such as the *do*-calculus are nonparametric, in the sense that they rely on conditional independence constraints only. More informative reductions are possible if one is willing to provide extra information, such as assuming linearity of causal effects. For such cases, other parametric constraints can be exploited (Spirtes et al. 2000; Pearl 2000).

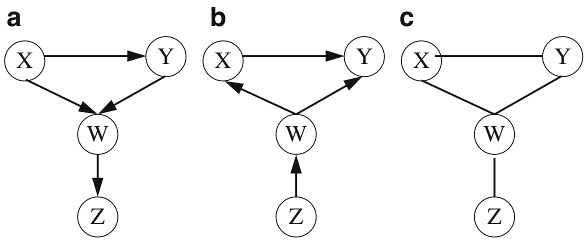
Learning Causal Structure

In all of the previous sections, we assumed that a causal graph was available. Since background knowledge is often limited, learning such graph structures becomes an important task. However, this cannot be accomplished without extra assumptions. To see why, consider again the example in Fig. 2a: if $a + bc = 0$, it follows that the X and Y are independent in the natural state. However, Y is *not* causally independent of X (if $b \neq 0$): $P(Y|do(X{=}x_1))$ and $P(Y|do(X{=}x_2))$ will be two different Gaussians with means $b \cdot x_1$ and $b \cdot x_2$, respectively.

This example demonstrates that an independence constraint that is testable by observational data does not warrant causal independence, at least based on the causal Markov condition only. However, an independence constraint that arises from particular identities such as $a + bc = 0$ is not *stable*, in the sense that it does not follow from the qualitative causal relations entailed by the Markov condition: a change in any of the parameter values will destroy such a constraint.

The artificiality of unstable independencies motivates an extra assumption: the *faithfulness* condition (Spirtes et al. 2000), also known as the *stability* condition (Pearl 2000). We say that a distribution P is faithful to a causal graph G if P is Markov with respect to G *and* if each conditional independence in P corresponds to some d-separation in G. That is, on top of the causal Markov condition, we assume that all independencies in P are entailed by the causal graph G.

The faithfulness condition allows us to reconstruct classes of causal graphs from observational data. In the simplest case, observing that X and Y are independent entails that there is no causal connection between X and Y. Conse-



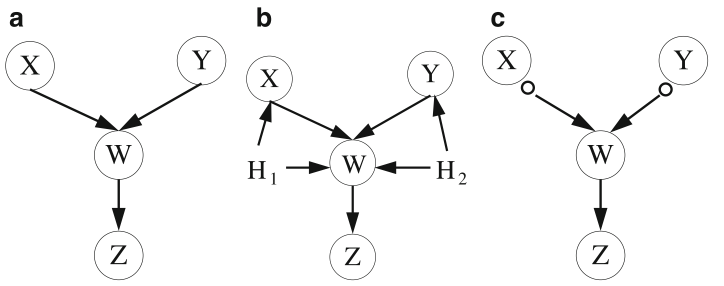
Causality, Fig. 3 (**a**) A particular causal graph which entails a few independence constraints, such as X and Z being independent given W. (**b**) A different causal graph that entails exactly the same independence constraints as in (**a**). (**c**) A representation for all graphs that entail the same conditional independencies as (**a**) and (**b**)

quently, $P(Y|do(X)) = P(Y|X) = P(Y)$. No interventional data was necessary to arrive at this conclusion, given the faithfulness condition.

In general, the solution is undetermined: more than one causal graph will be compatible with a set of observable independence constraints. Consider a simple example, where data is generated by a causal model with a causal graph given as in Fig. 3a. This graph entails some independencies, for instance, that X and Z are independent given W or that X and Y are not independent given any subset of $\{W, Z\}$. However, several other graphs entail the same conditional independences. The graph in Fig. 3b is one example. The learning task is then discovering an *equivalence class* of graphs, not necessarily a particular graph. This is in contrast with the problem of learning the structure of non-causal graphical models: the fact that there are other structures compatible with the data is not important in this case, since we will not use such graphical models to predict the effect of some hypothetical intervention. An equivalence class might not be enough information to reduce a desired causal query to a probabilistic query, but it might require much less prior knowledge than specifying a full causal graph.

Assume for now that no hidden common causes exist in this domain. In particular, the graphical object in Fig. 3c is a representation of the equivalence class of graphs that are compatible with the independencies encoded in Fig. 3a (Pearl 2000; Spirtes et al. 2000). All members of the equivalence class will have the same *skeleton* of this representation, i.e., the same adjacencies. An undirected edge indicates that there are two members in the equivalence class where directionality of this particular edge goes in opposite directions. Some different directions are illustrated in Fig. 3b. One can verify from the properties of d-separation that, if an expert or an experiment indicates that $X - W$ should be directed as $X \rightarrow W$, then the edge $W - Z$ is *compelled* to be directed as $W \rightarrow Z$: the direction $W \leftarrow Z$ is incompatible with the simultaneous findings that X and Z are independent given W and that X causes W.

More can be discovered if more independence constraints exist. In Fig. 4a, X is not a cause of Y. If we assume no hidden common causes exist in this domain, then no other causal graph is compatible with the independence constraints of Fig. 4a: the equivalence class is this graph only. However, the assumption of no hidden common causes is strong and undesirable. For instance, the graph in Fig. 4b, where H_1 and H_2 are hidden, is in the same equivalence class of (a). Yet, the graph in (a) indicates that $P(W|do(X)) = P(W|X)$, which can be arbitrarily different from the real $P(W|do(X))$ if Fig. 4b is the real graph. Some equivalence class representations, such as the Partial Ancestral Graph representation (Spirtes et al. 2000), are robust to hidden common causes: in Fig. 4c, an edge that has a circle as endpoint indicates that is not known if there is a causal path into both, e.g., X and W (which would be the case for a hidden common cause of X and W). The arrow into W does indicate that W cannot be a cause of X. A fully directed edge such as $W \rightarrow Z$ indicates total information: W is a cause of Z,



Causality, Fig. 4 (**a**) A particular causal graph with no other member on its equivalence class (assuming there are no hidden common causes). (**b**) Graph under the presence of two hidden common causes H_1 and H_2. (**c**) A representation for all graphs that entail the same conditional independencies as (**a**), without assuming the nonexistence of hidden common causes

Z is not a cause of W, and W and Z have no hidden common causes.

Given equivalence class representations and background knowledge, different types of algorithms explore independence constraints to learn an equivalence class. It is typically assumed that the true graph is acyclic. The basic structure is to evaluate how well a set of conditional independence hypotheses is supported by the data. Depending on which constraints are judged to hold in the population, we keep, delete, or orient edges accordingly. Some algorithms, such as the PC algorithm (Spirtes et al. 2000), test a single independence hypothesis at a time and assemble the individual outcomes in the end into an equivalence class representation. Other algorithms such as the GES algorithm (Meek 1997; Chickering 2002) start from a prior distribution for graphs and parameters and proceed to compare the marginal likelihood of members of different equivalence classes (which can be seen as a Bayesian joint test of independence hypotheses). In the end, this reduces to a search for the maximum a posteriori equivalence class estimator. Both PC and GES have consistency properties: in the limit of infinite data, they return the right equivalence class under the faithfulness assumption. However, both PC and GES, and most causal discovery algorithms, assume that there are no hidden common causes in the domain. The Fast Causal Inference (FCI) algorithm of Spirtes et al. (2000) is able to generate equivalence class representations as in Fig. 4c. As in the PC algorithm, this is done by testing a single independence hypothesis at a time and therefore is a high-variance estimator given small samples. A GES-like algorithm with the consistency properties of FCI is not currently known. An algorithm that allows for cyclic networks is discussed by Richardson (1996).

Semiparametric Models

Our examples relied on conditional independence constraints. In this case, the equivalence class is known as the *Markov equivalence class*. Markov equivalence classes are “nonparametric,” in the sense that they do not refer to any particular probability family. In practice, this advantage is limited by our ability to test independence hypotheses within flexible probability families. Another shortcoming of Markov equivalence classes is that they might be poorly informative if few independence constraints exist in the population. This will happen, for instance, if a single hidden variable is a common cause of all observed variables. If one is willing to incorporate further assumptions, such as linearity of causal relationships, semiparametric constraints can be used to define other types of equivalence classes that are more discriminative than the Markov equivalence class. Silva et al. (2006) describe how some rank constraints in the covariance matrix of the observed variables can be used to learn the structure of linear models, even if no independence constraints are observable. Shimizu et al. (2006) provide a solution to find the causal ordering of a linear DAG model without latent variables, by exploiting information beyond the second moments of a distribution in the non-Gaussian case. Entner et al. (2012) introduce an approach to estimate causal effects in non-Gaussian linear systems under some assumptions of directionality but allowing for unmeasured confounding. Peters et al. (2014) develop a general method for learning directionality in nonlinear models with additive noise.

Confidence Intervals

Several causal learning algorithms such as the PC and FCI algorithms (Spirtes et al. 2000) are consistent, in the sense that they can recover the correct equivalence class given the faithfulness assumption and an infinite amount of data. Although point estimates of causal effects are important, it is also important to provide confidence intervals. From a frequentist perspective, it has been shown that this is not possible given the faithfulness assumption only (Robins et al. 2003). An intuitive explanation is as follows: consider the model such as the one in Fig. 2a. For any given sample size, there is at least one model such that the associations due to the paths $X \leftarrow H \rightarrow Y$ and $X \rightarrow Y$ nearly cancel each other (faithfulness is still preserved), making the covariance of X and Y statistically indistinguishable from zero. In order to achieve uniform consistency, causal inference algorithms need assumptions stronger than faithfulness. Zhang and Spirtes (2003) provide some directions.

Other Languages and Tasks in Causal Learning

A closely related language for representing causal models is the *potential outcomes* framework popularized by Donald Rubin (Rubin 2005). In this case, random variables for a same variable Y are defined for each possible state of the intervened variable X. Notice that, by definition, only one of the possible Y outcomes can be observed for any specific data point. This framework is popular in the statistics literature as a type of missing data model. The relation between potential outcomes and several other representations of causality is discussed by Richardson and Robins (2013).

A case where potential outcomes become particularly motivated is in *causal explanation*. In this setup, the model is asked for the probability that a particular event in time was the cause of a particular outcome. This is often cast as a *counterfactual question*: had A been false, would B still have happened? Questions in history and law are of this type: the legal responsibility of an airplane manufacturer in an accident depends on technical malfunction being an *actual cause* of the accident. Ultimately, such issues of causal explanation, actual causation and other counterfactual answers, are untestable. Although machine learning can be a useful tool to derive the consequences of assumptions combined with data about other events of the same type, in general the answers will not be robust to changes in the assumptions, and the proper assumptions ultimately cannot be selected with the available data. Some advances in generating explanations with causal models are described by Halpern and Pearl (2005).

Recommended Reading

Chickering D (2002) Optimal structure identification with greedy search. J Mach Learn Res 3:507–554

Cooper G, Yoo C (1999) Causal discovery from a mixture of experimental and observational data. In: Proceedings of the 15th conference on uncertainty in artificial intelligencem (UAI-1999), Stockholm, pp 116–125

Dawid AP (2003) Causal inference using influence diagrams: the problem of partial compliance. In: Green PJ, Hjort NL, Richardson S (eds) Highly structured stochastic systems. Oxford University Press, New York, pp 45–65

Eaton D, Murphy K (2007) Exact Bayesian structure learning from uncertain interventions. In: Proceedings of the 11th international conference on artificial intelligence and statistics (AISTATS-2007), San Juan, pp 107–114

Entner D, Hoyer PO, Spirtes P (2012) Statistical test for consistent estimation of causal effects in linear non-gaussian models. In: Proceedings of the 15th international conference on artificial intelligence and statistics (AISTATS-2012), La Palma, pp 364–372

Halpern J, Pearl J (2005) Causes and explanations: a structural-model approach. Part II: explanations. Br J Philos Sci 56:889–911
Hyttinen A, Eberhardt F, Hoyer PO (2013) Experiment selection for causal discovery. J Mach Learn Res 14:3041–3071
Meek C (1997) Graphical models: selecting causal and statistical models. PhD thesis, Carnegie Mellon University
Pearl J (2000) Causality: models, reasoning and inference. Cambridge University Press, New York
Peters J, Mooij JM, Janzing D, Schölkopf B (2014) Causal discovery with continuous additive noise models. J Mach Learn Res 15:2009–2053
Richardson TS (1996) A discovery algorithm for directed cyclic graphs. In: Proceedings of 12th conference on uncertainty in artificial intelligence, Portland
Richardson TS, Robins J (2013) Single world intervention graphs (SWIGs): a unification of the counterfactual and graphical approaches to causality. Working Paper Number 128, Center for Statistics and the Social Sciences, University of Washington
Robins J, Scheines R, Spirtes P, Wasserman L (2003) Uniform consistency in causal inference. Biometrika 90:491–515
Rosenbaum P (2002) Observational studies. Springer, New York
Rubin D (2005) Causal inference using potential outcomes: design, modeling, decisions. J Am Stat Assoc 100(469):322–331
Sachs K, Perez O, Pe'er D, Lauffenburger D, Nolan G (2005) Causal protein-signaling networks derived from multiparameter single-cell data. Science 308:523–529
Shimizu S, Hoyer P, Hyvärinen A, Kerminen A (2006) A linear non-Gaussian acyclic model for causal discovery. J Mach Learn Res 7:2003–2030
Silva R, Scheines R, Glymour C, Spirtes P (2006) Learning the structure of linear latent variable models. J Mach Lear Res 7:191–246
Spirtes P, Glymour C, Scheines R (2000) Causation, prediction and search. MIT Press, Cambridge
Wasserman L (2004) All of statistics. Springer, New York
Zhang J, Spirtes P (2003) Strong faithfulness and uniform consistency in causal inference. In: Proceedings of the 19th conference in uncertainty in artificial intelligence (UAI-2013), Acapulco, pp 632–639

CC

▶ Cascade Correlation

Certainty Equivalence Principle

▶ Internal Model Control

Characteristic

▶ Attribute

Citation or Reference Matching (When Applied to Bibliographic Data)

▶ Record Linkage

City Block Distance

▶ Manhattan Distance

Class

Chris Drummond
National Research Council of Canada, Ottawa, ON, Canada

Synonyms

Category; Collection; Kind; Set; Sort; Type

Definition

A class is a collection of things that might reasonably be grouped together. Classes that we commonly encounter have simple names so, as humans, we can easily refer to them. The class of dogs, for example, allows me to say "my dog ate my newspaper" without having to describe a particular dog, or indeed, a particular newspaper. In machine learning, the name of the class is called the class label. Exactly what it means to belong to a class, or category, is a complex philosophical question but often we think of a class in terms of the common properties of its members. We think particularly of those properties which seperate them from other things which are in many ways similar, e.g., cats mieow and dogs bow-wow. We would be unlikely to form a class from a random collection of things, as they would share no common properties. Knowing something belonged to such a collection would be of no particular benefit. Although many every day classes will have simple names, we may construct them however we like, e.g., "The things I like to eat for breakfast on a Saturday morning." As there is no simple name for such a collection, in machine learning we would typically refer to it as the positive class, and all occurences of it are positive examples; the negative class would be everything else.

Motivation and Background

The idea of a class is important in learning. If we discover something belongs to a class, we suddenly know quite a lot about it even if we have not encountered that particular example before. In machine learning, our use of the term accords closely with the mathematical definition of a class, as a collection of sets unambiguously defined by a property that all its members share. It also accords with the idea of equivalence classes, which group similar things. Sets have an intension, the description of what it means to be a member, and an extension, things that belong to the set, useful properties of a class in machine learning. Class is also a term used extensively in knowledge bases to denote an important relationship between groups, of sub-class and super class. Learning is often viewed as a way of solving the knowledge acquisition bottleneck (Buchanan et al. 1983) in knowledge bases and the use of the term class in machine learning highlights this connection.

Recommended Reading

Buchanan B, Barstow D, Bechtel R, Bennett J, Clancey W, Kulikowski C et al (1983) Constructing an expert system. In: Hayes-Roth F, Waterman DA, Lenat DB (eds) Building expert systems. Addison-Wesley, Reading, pp 127–167

Class Binarization

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

Abstract

Many learning algorithms are only designed to separate two classes from each other. For example, ▸ concept-learning algorithms assume positive examples and negative examples (counterexamples) for the concept to learn, and many statistical learning techniques, such as neural networks or ▸ support vector machines, can only find a single separating decision surface. One way to apply these algorithms to multi-class problem is to transform the original multi-class problem into multiple binary problems.

Synonyms

Error-correcting output codes (ECOC); One-against-all training; One-against-one training; Pairwise classification

Methods

The best-known techniques are:

One against all: one concept-learning problem is defined for each class, i.e., each class is in turn used as the positive class, and all other classes form the negative class.

Pairwise (One against one): one concept is learned for each pair of classes (Fürnkranz 2002). This may be viewed as a special case of ▸ preference learning.

Error-correcting output codes: ECOC allow arbitrary subsets of the classes to form the positive and negative classes of the binary problems. In the original formulation (Dietterich and Bakiri 1995), all classes have to be used for each problem, a later generalization (Allwein et al. 2000) allows arbitrary combinations. Clearly, one against all and one against one are special cases of ECOC.

The predictions of the binary classifiers must then be combined into an overall prediction. Commonly used techniques include voting and finding the nearest neighbor in the ECOC decoding matrix (Allwein et al. 2000).

Cross-References

▶ Preference Learning
▶ Rule Learning

Recommended Reading

Allwein EL, Schapire RE, Singer Y (2000) Reducing multiclass to binary: a unifying approach for margin classifiers. J Mach Learn Res 1: 113–141
Dietterich TG, Bakiri G (1995) Solving multiclass learning problems via error-correcting output codes. J Artif Intell Res 2:263–286
Fürnkranz J (2002) Round robin classification. J Mach Learn Res 2:721–747. http://www.ai.mit.edu/projects/jmlr/papers/volume2/fuernkranz02a/html/.

Class Imbalance Problem

Charles X. Ling and Victor S. Sheng
The University of Western Ontario, London, ON, Canada

Definition

Data are said to suffer the *Class Imbalance Problem* when the class distributions are highly imbalanced. In this context, many ▶ classification learning algorithms have low predictive accuracy for the infrequent class. ▶ Cost-sensitive learning is a common approach to solve this problem.

Motivation and Background

Class imbalanced datasets occur in many real-world applications where the class distributions of data are highly imbalanced. For the two-class case, without loss of generality, one assumes that the minority or rare class is the positive class, and the majority class is the negative class. Often the minority class is very infrequent, such as 1 % of the dataset. If one applies most traditional (cost-insensitive) classifiers on the dataset, they are likely to predict everything as negative (the majority class). This was often regarded as a problem in learning from highly imbalanced datasets.

However, Provost (2000) describes two fundamental assumptions that are often made by traditional cost-insensitive classifiers. The first is that the goal of the classifiers is to maximize the accuracy (or minimize the error rate); the second is that the class distribution of the training and test datasets is the same. Under these two assumptions, predicting everything as negative for a highly imbalanced dataset *is often the right thing to do*. Drummond and Holte (2005) show that it is usually very difficult to outperform this simple classifier in this situation.

Thus, the imbalanced class problem becomes meaningful only if one or both of the two assumptions above are not true; that is, if the cost of different types of error (false positive and false negative in the binary classification) is not the same, or if the class distribution in the test data is different from that of the training data. The first case can be dealt with effectively using methods in cost-sensitive meta-learning (see ▶ Cost-sensitive learning).

In the case when the misclassification cost is not equal, it is usually more expensive to misclassify a minority (positive) example into the majority (negative) class, than a majority example into the minority class (otherwise it is more plausible to predict everything as negative). That is, *FNcost* > *FPcost*. Thus, given the values of *FNcost* and *FPcost*, a variety of cost-sensitive meta-learning methods can be, and have been, used to solve the class imbalance problem (Japkowicz and Stephen 2002; Ling and Li 1998). If the values of *FNcost* and *FPcost* are not unknown explicitly, *FNcost* and *FPcost* can be assigned to be proportional to the number of positive and negative training cases (Japkowicz and Stephen 2002).

In case the class distributions of training and test datasets are different (e.g., if the training data is highly imbalanced but the test data is more balanced), an obvious approach is to sample the training data such that its class distribution is the same as the test data. This can be achieved by oversampling (creating multiple copies of examples of) the minority class and/or undersampling (selecting a subset of) the majority class (Provost 2000).

Note that sometimes the number of examples of the minority class is too small for classifiers to learn adequately. This is the problem of insufficient (small) training data and different from that of imbalanced datasets.

Recommended Reading

Drummond C, Holte R (2000) Exploiting the cost (in)sensitivity of decision tree splitting criteria. In: Proceedings of the seventeenth international conference on machine learning, Stanford, pp 239–246

Drummond C, Holte R (2005) Severe class imbalance: why better algorithms aren't the answer. In: Proceedings of the sixteenth European conference of machine learning, Porto, vol 3720. LNAI, pp 539–546

Japkowicz N, Stephen S (2002) The class imbalance problem: a systematic study. Intell Data Anal 6(5):429–450

Ling CX, Li C (1998) Data mining for direct marketing – specific problems and solutions. In: Proceedings of fourth international conference on knowledge discovery and data mining (KDD-98), New York City, pp 73–79

Provost F (2000) Machine learning from imbalanced data sets 101. In: Proceedings of the AAAI'2000 workshop on imbalanced data

Classification

Chris Drummond
National Research Council of Canada, Ottawa, ON, Canada

Synonyms

Categorization; Generalization; Identification; Induction; Recognition

Definition

In common usage, the word classification means to put things into categories, group them together in some useful way. If we are screening for a disease, we would group people into those with the disease and those without. We, as humans, usually do this because things in a group, called a ▸ class in machine learning, share common characteristics. If we know the class of something, we know a lot about it. In machine learning, the term classification is most commonly associated with a particular type of learning where examples of one or more ▸ classes, labeled with the name of the class, are given to the learning algorithm. The algorithm produces a classifier which maps the properties of these examples, normally expressed as ▸ attribute-value pairs, to the class labels. A new example whose class is unknown is classified when it is given a class label by the classifier based on its properties. In machine learning, we use the word classification because we call the grouping of things a class. We should note, however, that other fields use different terms. In philosophy and statistics, the term categorization is more commonly used. In many areas, in fact, classification often refers to what is called clustering in machines learning.

Motivation and Background

Classification is a common, and important, human activity. Knowing something's class allows us to predict many of its properties and so act appropriately. Telling other people its class allows them to do the same, making for efficient communication. This emphasizes two commonly held views of the objectives of learning. First, it is a means of ▸ generalization, to predict accurately the values for previously unseen examples. Second, it is a means of compression, to make transmission or communication more efficient. Classification is certainly not a new idea and has been studied for some considerable time. From the days of the early Greek philosophers such as Socrates, we had the idea of categorization. There are essential properties of things that make them what they are. It embodies the idea that there are natural kinds, ways of grouping things, that are inherent in the world. A major goal of learning, therefore, is recognizing natural kinds, establishing the necessary and sufficient conditions for belonging to a category. This "classical" view of categorization, most often attributed to Aristotle, is now strongly disputed. The main competitor is prototype theory; things are categorized by their similarity to a prototypical example (Lakoff 1987), either real or imagined. There is also much debate in psychology (Ashby and Maddox 2005), where many argue that there is no single method of categorization used by humans.

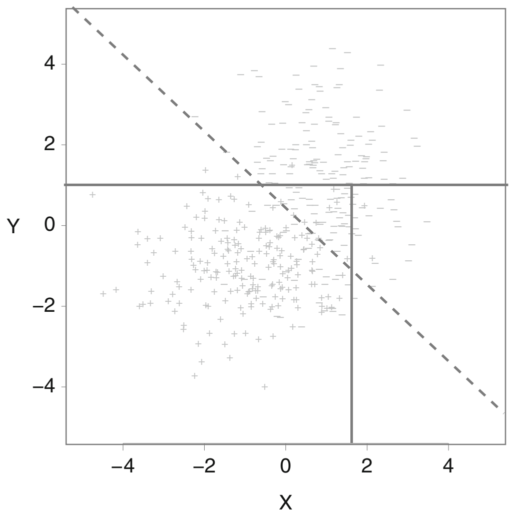
As much of the inspiration for machine learning originated in how humans learn, it is unsurprising that our algorithms reflect these distinctions. ▶ Nearest neighbor algorithms would seem to have much in common with prototype theory. These have been part of pattern recognition for some time (Cover and Hart 1967) and have become popular in machine learning, more recently, as ▶ instance-based learners (Aha et al. 1991). In machine learning, we measure the distance to one or more members of a concept rather a specially constructed prototype. So, this type of learning is perhaps more a case of the exemplar learning found in the psychological literature, where multiple examples represent a category. The closest we have to prototype learning occurs in clustering, a type of ▶ unsupervised learning, rather than classification. For example, in ▶ k-means clustering group membership is determined by closeness to a central value.

In the early days of machine learning, our algorithms (Mitchell 1977; Winston 1975) had much in common with the classical theory of categorization in philosophy and psychology. It was assumed that the data were consistent, there were no examples with the same attribute values but belonging to different classes. It was quickly realized that, even if the properties where necessary and sufficient to capture the class, there was often noise in the attribute and perhaps the class values. So, complete consistency was seldom attainable in practice. New ▶ classification algorithms were designed, which could tolerate some noise, such as ▶ decision trees (Breiman et al. 1984; Quinlan 1986, 1993) and rule-based learners (see ▶ Rule Learning) (Clark and Niblett 1989; Holte 1993; Michalski 1983).

Structure of the Learning System

Whether one uses instance-based learning, rule-based learning, decision trees, or indeed any other classification algorithm, the end result is the division of the input space into regions belonging to a single class. The input space is defined by the Cartesian product of the attributes, all possible combinations of possible values.

As a simple example, Fig. 1 shows two classes $+$ and $-$, each a random sample of a normal distribution. The attributes are X and Y of real type. The values for each attribute range from $\pm\infty$. The figure shows a couple of alternative ways that the space may be divided into regions. The bold dark lines, construct regions using lines that are parallel to the axes. New examples that have Y less than 1 and X less than 1.5 with be classified as $+$, all others classified as $-$. Decision trees and rules form this type of boundary. A ▶ linear discriminant function, such as the bold dashed line, would divide the space into half-spaces, with new examples below the line being classified as $+$ and those above as $-$. Instance-based learning will also divide the space into re-



Classification, Fig. 1 Dividing the input space gions but the boundary is implicit. Classification occurs by choosing the class of the majority of the nearest neighbors to a new example. To make the boundary explicit, we could mark the regions where an example would be classified as $+$ and those classified as $-$. We would end up with regions bounded by polygons.

What differs among the algorithms is the shape of the regions, and how and when they are chosen. Sometimes the regions are implicit as in lazy learners (see ▸ Lazy Learning) (Aha 1997), where the boundaries are not decided until a new example is being classified. Sometimes the regions are determined by decision theory as in generative classifiers (see ▸ Generative Learning) (Rubinstein and Hastie 1997), which model the full joint distribution of the classes. For all classifiers though, the input space is effectively partitioned into regions representing a single class.

Applications

One of the reasons that classification is an important part of machine learning is that it has proved to be a very useful technique for solving practical problems. Classification has been used to help scientists in the exploration, and comprehension, of their particular domains of interest. It has also been used to help solve significant industrial problems. Over the years a number of authors have stressed the importance of applications to machine learning and listed many successful examples (Brachman et al. 1996; Langley and Simon 1995; Michie 1982). There have also been workshops on applications (Kodratoff 1994; Aha and Riddle 1995; Engels et al. 1997) at major machine learning conferences and a special issue of Machine Learning (Kohavi and Provost 1998), one of the main journals in the field. There are now conferences that are highly focused on applications. Collocated with major artificial intelligence conferences is the Innovative Applications of Artificial Intelligence conference. Since 1989, this conference has highlighted practical applications of machine learning, including classification (Schorr and Rappaport 1989). In addition, there are now at least two major knowledge discovery and data mining conferences (Fayyad and Uthurusamy 1995; Komorowski and Zytkow 1997) with a strong focus on applications.

Future Directions

In machine learning, there are already a large number of different classification algorithms, yet new ones still appear. It seems unlikely that there is an end in sight. The "no free lunch theory" (Wolpert and Macready 1997) indicates that there will never be a single best algorithm, better than all others in terms of predictive power. However, apart from their predictive performance, each classifier has its own attractive properties which are important to different groups of people. So, new algorithms are still of value. Further, even if we are solely concerned about performance, it may be useful to have many different algorithms, all with their own biases (see ▸ Inductive Bias). They may be combined together to form an ensemble classifier (Caruana et al. 2004), which outperforms single classifiers of one type (see ▸ Ensemble Learning).

Limitations

Classification has been critical part of machine research for some time. There is a concern that the emphasis on classification, and more generally on ▸ supervised learning, is too strong. Certainly much of human learning does not use, or require, labels supplied by an expert. Arguably, unsupervised learning should play a more central role in machine learning research. Although classification does require a label, it does necessarily need an expert to provide labeled examples. Many successful applications rely on finding some, easily identifiable, property which stands in for the class.

Recommended Reading

Aha DW (1997) Editorial. Artif Intell Rev 11(1–5):1–6
Aha DW, Riddle PJ (eds)(1995) Workshop on applying machine learning in practice. In: Proceedings of the

12th international conference on machine learning, Tahoe City
Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. Mach Learn 6(1):37–66
Ashby FG, Maddox WT (2005) Human category learning. Ann Rev Psychol 56:149–178
Bishop CM (2007) Pattern recognition and machine learning. Springer, New York
Brachman RJ, Khabaza T, Kloesgen W, Piatetsky-Shapiro G, Simoudis E (1996) Mining business databases. Commun ACM 39(11):42–48
Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth, Belmont
Caruana R, Niculescu-Mizil A, Crew G, Ksikes A (2004) Ensemble selection from libraries of models. In: Proceedings of the 21st international conference on machine learning, Banff, pp 137–144
Clark P, Niblett T (1989) The CN2 induction algorithm. Mach Learn 3:261–284
Cover T, Hart P (1967) Nearest neighbor pattern classification. IEEE Trans Inf Theory 13:21–27
Dietterich T, Shavlik J (eds) Readings in machine learning. Morgan Kaufmann, San Mateo
Engels R, Evans B, Herrmann J, Verdenius F (eds) (1997) Workshop on machine learning applications in the real world; methodological aspects and implications. In: Proceedings of the 14th international conference on machine learning, Nashville
Fayyad UM, Uthurusamy R (eds)(1995) Proceedings of the first international conference on knowledge discovery and data mining, Montreal
Holte RC (1993) Very simple classification rules perform well on most commonly used datasets. Mach Learn 11(1):63–91
Kodratoff Y (ed)(1994) Proceedings of MLNet workshop on industrial application of machine learning, Douran
Kodratoff Y, Michalski RS (1990) Machine learning: an artificial intelligence approach, vol 3. Morgan Kaufmann, San Mateo
Kohavi R, Provost F (1998) Glossary of terms. Editorial for the special issue on applications of machine learning and the knowledge discovery process. Mach Learn 30(2/3)
Komorowski HJ, Zytkow JM (eds) (1997) Proceedings of the first European conference on principles of data mining and knowledge discovery
Lakoff G (1987) Women, fire and dangerous things. University of Chicago Press, Chicago
Langley P, Simon HA (1995) Applications of machine learning and rule induction. Commun ACM 38(11):54–64
Michalski RS (1983) A theory and methodology of inductive learning. In: Michalski RS, Carbonell TJ, Mitchell TM (eds) Machine learning: an artificial intelligence approach. TIOGA Publishing, Palo Alto, pp 83–134
Michalski RS, Carbonell JG, Mitchell TM (eds) (1983) Machine learning: an artificial intelligence approach. Tioga Publishing Company, Palo Alto
Michalski RS, Carbonell JG, Mitchell TM (eds) (1986) Machine learning: an artificial intelligence approach, vol 2. Morgan Kaufmann, San Mateo
Michie D (1982) Machine intelligence and related topics. Gordon and Breach Science Publishers, New York
Mitchell TM (1977) Version spaces: a candidate elimination approach to rule learning. In: Proceedings of the fifth international joint conferences on artificial intelligence, Cambridge, pp 305–310
Mitchell TM (1997) Machine learning. McGraw-Hill, Boston
Quinlan JR (1986) Induction of decision trees. Mach Learn 1:81–106
Quinlan JR (1993) C4.5 programs for machine learning. Morgan Kaufmann, San Mateo
Rubinstein YD, Hastie T (1997) Discriminative vs informative learning. In: Proceedings of the third international conference on knowledge discovery and data mining, Newport Beach, pp 49–53
Russell S, Norvig P (2003) Artificial intelligence: a modern approach. Prentice-Hall, Upper Saddle River
Schorr H, Rappaport A (eds) (1989) Proceedings of the first conference on innovative applications of artificial intelligence, Stanford
Winston PH (1975) Learning structural descriptions from examples. In: Winston PH (ed) The psychology of computer vision. McGraw-Hill, New York, pp 157–209
Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, San Fransisco
Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82

Classification Algorithms

There is a very large number of classification algorithms, including ▶ decision trees, ▶ instance-based learners, ▶ support vector machines, ▶ rule-based learners, ▶ neural networks, ▶ Bayesian networks. There also ways of combining them into ensemble classifiers such as ▶ boosting, ▶ bagging, ▶ stacking, and forests of trees.

To delve deeper into classifiers and their role in machine learning, a number of books are recommended covering machine learning (Bishop 2007; Mitchell 1997; Witten and Frank 2005) and artificial intelligence (Russell and Norvig 2003) in general. Seminal papers on classifiers can be found in collections of papers on machine learning (Dietterich and Shavlik 1990; Kodratoff and Michalski 1990; Michalski et al. 1983, 1986).

Recommended Reading

Bishop CM (2007) Pattern recognition and machine learning. Springer, New York
Dietterich T, Shavlik J (eds) (1990) Readings in machine learning. Morgan Kaufmann, San Mateo
Kodratoff Y, Michalski RS (1990) Machine learning: an artificial intelligence approach, vol 3. Morgan Kaufmann, San Mateo
Michalski RS, Carbonell JG, Mitchell TM (eds) (1983) Machine learning: an artificial intelligence approach. Tioga Publishing Company, Palo Alto
Michalski RS, Carbonell JG, Mitchell TM (eds) (1986) Machine learning: an artificial intelligence approach, vol 2. Morgan Kaufmann, San Mateo
Mitchell TM (1997) Machine learning. McGraw-Hill, Boston
Russell S, Norvig P (2003) Artificial intelligence: a modern approach. Prentice-Hall, Upper Saddle River
Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, San Fransisco

Classification Learning

▶ Concept Learning

Classification Rule

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

Abstract

A classification rule is an `IF-THEN` rule. The condition of the rule (the *rule body* or *antecedent*) typically consists of a conjunction of Boolean terms, each one constituting a constraint that needs to be satisfied by an example. If all constraints are satisfied, the rule is said to *fire*, and the example is said to be *covered* by the rule. The *rule head* (also called the *consequent* or *conclusion*) consists of a single ▶ class value, which is predicted in case the rule fires. This is in contrast to ▶ association rules, which allow multiple features in the head.

Method

Typical terms consist of tests for the presence of a particular ▶ attribute value or, in the case of numerical attributes, of an inequality that requires that the observed value is above or below a threshold. More expressive constraints include *set-valued attributes* (several values of the same attribute can be observed in the training examples), *internal disjunctions* (only one of several values of the same attribute needs to present), *hierarchical attributes* (certain values of the attributes subsume other values), etc.

Conjunctive combinations of features may be viewed as statements in ▶ propositional logic (▶ propositional rules). If relations between features can be considered (i.e., if propositions can be formulated in ▶ first-order logic), we speak of ▶ first-order rules.

Cross-References

▶ Association Rule
▶ Decision List
▶ First-Order Logic
▶ Propositional Logic
▶ Rule Learning

Classification Tree

▶ Decision Tree

Classifier Calibration

Peter A. Flach
Department of Computer Science, University of Bristol, Bristol, UK

Abstract

Classifier calibration is concerned with the scale on which a classifier's scores are expressed. While a classifier ultimately maps instances to discrete classes, it is often beneficial to decompose this mapping into a *scoring classifier* which outputs one or more real-valued numbers and a *decision rule* which converts these numbers into predicted classes. For example, a linear classifier might output a positive or negative score whose magnitude is proportional to the distance between the instance and the *decision boundary*, in which case the decision rule would be a simple threshold on that score. The advantage of calibrating these scores to a known, domain-independent scale is that the decision rule then also takes a domain-independent form and does not have to be learned. The best-known example of this occurs when the classifier's scores approximate, in a precise sense, the ▸ posterior probability over the classes; the main advantage of this is that the optimal decision rule is to predict the class that minimizes expected cost averaged over all possible true classes. The main methods to obtain calibrated scores are *logistic calibration*, which is a parametric method that assumes that the distances on either side of the decision boundary are normally distributed and a nonparametric alternative that is variously known as *isotonic regression*, the *pool adjacent violators* (PAV) method or the *ROC convex hull* (ROCCH) method.

Synonyms

Isotonic calibration; Logistic calibration; Probability calibration; Sigmoid calibration

Motivation and Background

A predictive model can be said to be well calibrated if its predictions match observed distributions in the data. In particular, a probabilistic classifier is well calibrated if, among the instances receiving a predicted probability vector p, the class distribution is approximately distributed as p. Hence, the classifier approximates, in some sense, the class posterior, although the approximation can be crude: for example, a constant classifier predicting the overall class distribution for every instance is perfectly calibrated in this sense.

Calibration is closely related to optimal decision making and cost-sensitive classification, where we wish to determine the predicted class that minimizes expected misclassification cost averaged over all possible true classes. The better our estimates of the class posterior are, the closer we get to the (irreducible) Bayes risk. A sufficiently calibrated classifier can be simply thresholded at a threshold directly derived from the misclassification costs. Similar thresholds can be used to optimally adapt to a change in class prior or to a combination of both.

Some training algorithms naturally yield well-calibrated classifiers, including ▸ logistic regression and ▸ decision trees (with Laplace smoothing but without pruning; Provost and Domingos 2003; Ferri et al. 2003). Others do not take sufficient account of distributional factors (e.g., ▸ support vector machines) or make unrealistic assumptions (e.g., ▸ naive Bayes) and need to be calibrated in post-processing. Well-established calibration methods include logistic calibration (parametric) and the ROC convex hull method, also known as pair-adjacent violators and isotonic regression (nonparametric).

In order to evaluate the quality of probability estimates, we can use the Brier score, which measures mean squared deviation from the "ideal" probabilities 1 and 0. The Brier score can be decomposed into refinement loss and calibration loss, where the former assesses the likelihood of instances from different classes receiving the same probability estimate, and the latter measures the mean squared deviation from the empirical probabilities. Both quantities can be further decomposed: e.g., the refinement loss has model-dependent and model-independent components (grouping loss and irreducible loss), and the calibration loss has a component that can be reduced to zero by a simple affine transformation of the scores (Kull and Flach 2015). These losses and decompositions can be visualized by means of various cost curves.

One topic of current interest is that the above view of classifier calibration is closely tied to a particular family of losses (error rate and cost-sensitive variants). Class posteriors are the right thing to model for this family, but not for others. For example, it is known that thresholds on the class posterior that are optimal for F1-score are lower than optimal thresholds for accuracy, but there is no one-to-one mapping between the two. The cost-sensitive perspective offers an alternative view of calibration: a classifier is well calibrated if it outputs, for every instance, the cost parameter for which that instance is on (or close to) the decision boundary. This naturally leads to the idea of classifiers outputting several scores, each calibrated for a different cost model (e.g., accuracy and F1-score).

Solutions

We start by demonstrating that better approximation of posterior probabilities allows the classifier to make better predictions in a decision-theoretic sense (see also ▸Cost-Sensitive Classification). Denote the cost of predicting class j for an instance of true class i as $C(\hat{Y}=j|Y=i)$. Since we don't know the true class of an unlabelled instance, we need to base our prediction on an assessment of the expected cost over all possible true classes. The (true) expected cost of predicting class j for instance x is

$$C(\hat{Y}=j|X=x) = \sum_i P(Y=i|X=x)C(\hat{Y}=j|Y=i) \quad (1)$$

where $P(Y=i|X=x)$ is the probability of instance x having true class i. The optimal decision is then to predict the class which minimizes expected cost:

$$\begin{aligned}\hat{Y}^* &= \arg\min_j C(\hat{Y}=j|X=x)\\ &= \arg\min_j \sum_i P(Y=i|X=x)C(\hat{Y}=j|Y=i)\end{aligned}$$

In the special case that all misclassifications have equal cost, we can assume without loss of generality that $C(\hat{Y}=j|Y=i)=1$ for $i\neq j$ and $C(\hat{Y}=j|Y=i)=0$ for $i=j$, which gives

$$\begin{aligned}C(\hat{Y}=j|X=x) &= \sum_{i\neq j} P(Y=i|X=x)\\ &= 1-P(Y=j|X=x)\\ \hat{Y}^* &= \arg\min_j [1-P(Y=j|X=x)]\\ &= \arg\max_j P(Y=j|X=x)\end{aligned}$$

The main point is that *knowing the true class posterior allows the classifier to make optimal decisions*, either in a cost-sensitive or cost-indifferent setting. It therefore makes sense for a classifier to (approximately) learn the true class posterior. (Notice that this crucially depends on the cost model expressed by Eq. (1), which assumes that our aim is to optimize a cost-based version of accuracy. A different cost model will require a different notion of calibrated score, as will be briefly considered later for the case of the F_β score.) For the remainder of this entry, we will concentrate on binary classification, returning to the challenges of multiclass calibration at the end.

Optimal Decision Thresholds

In binary classification we have the following expected costs for positive and negative predictions:

$$\begin{aligned}C(\hat{Y}=+|X=x) &= P(+|x)C(+|+)\\ &\quad + (1-P(+|x))C(+|-)\\ C(\hat{Y}=-|X=x) &= P(+|x)C(-|+)\\ &\quad + (1-P(+|x))C(-|-)\end{aligned}$$

where $P(+|x)$ is shorthand for $P(Y{=}+|X{=}x)$ and $C(j|i)$ for $C(\hat{Y} = j|Y = i)$. On the optimal decision boundary, these two expected costs are equal, which gives

$$\begin{aligned}&P(+|x)C(+|+) + (1 - P(+|x))C(+|-)\\&\quad = P(+|x)C(-|+) + (1 - P(+|x))C(-|-)\end{aligned}$$

and so

$$P(+|x) = \frac{C(+|-) - C(-|-)}{C(+|-) - C(-|-) + C(-|+) - C(+|+)} \triangleq c \tag{2}$$

This demonstrates that, from a decision-theoretic perspective, the cost matrix has one degree of freedom. Without loss of generality, we can therefore assume that costs are expressed on an arbitrary (but linear) scale and that correct classifications have zero cost: under this interpretation, c quantifies the proportion of loss attributed to the negatives if equal numbers of positives and negatives are misclassified. This *relative cost* then gives the optimal threshold on the positive posterior. For example, if a false positive incurs four units of cost and a false negative one unit, then $c = 4/5$, and hence we would increase the decision threshold from the default, cost-indifferent threshold of $1/2$, in order to make fewer positive predictions which are much more costly when wrong.

How is the class posterior affected when the class prior changes, but the class-conditional likelihoods $P(X|Y)$ stay the same? Suppose the proportion of positives changes from π to π', and let p denote the posterior probability under prior π, then Elkan (2001) derives the following expression for the posterior probability under prior π' (assuming the class-conditional likelihoods remain unchanged):

$$p' = \pi' \frac{p - p\pi}{\pi - p\pi + \pi' p - \pi\pi'}$$

Noting that the denominator can be rewritten to $(1 - \pi)\pi' p + \pi(1 - \pi')(1 - p)$ and switching to odds, we obtain an expression that can be rewritten as a product of odds:

$$\frac{p'}{1 - p'} = \frac{\pi'}{1 - \pi'} \frac{1 - \pi}{\pi} \frac{p}{1 - p}$$

This is best interpreted right to left. The rightmost term is the posterior odds under the original prior π; multiplying this with the reciprocal of the original prior odds gives the likelihood ratio, and multiplying this again with the new prior odds gives the desired posterior odds under the new prior π'. For example, if at training time we have balanced classes ($\pi = 1/2$), while at deployment time we have 20 % positives ($\pi' = 1/5$), then p' is adjusted downward accordingly. Conversely, the (cost-indifferent) deployment decision threshold $p' = 1/2$ corresponds to $p = 4/5$, highlighting the duality with the cost-sensitive example above.

More generally, for $p' = 1/2$ we have that

$$p = \frac{(1 - \pi')\pi}{(1 - \pi')\pi + \pi'(1 - \pi)} \triangleq d \tag{3}$$

is the decision threshold on the original posterior that takes account of the changed class distribution. Hence, d parameterizes the distribution change from π to π' in the same way as c parameterizes the change from cost-indifference to cost-sensitivity. Clearly, this opens up the way to combining changes in both class and cost distribution in a straightforward way.

Evaluation Metrics for Calibration

A multiclass scoring classifier outputs, for every test instance x, a probability vector $(p_1(x), \ldots, p_k(x))$, where k is the number of classes and $\sum_{i=1}^{k} p_i(x) = 1$. Suppose the true class is represented by a bit vector $(b_1(x), \ldots, b_k(x))$ such that the bit corresponding to the true class is set to 1 and the remaining are 0, then the *Brier score* over a test set T is defined as

$$BS = \frac{1}{|T|} \sum_{x \in T} \frac{1}{2} \sum_{i=1}^{k} (p_i(x) - b_i(x))^2$$

The factor 1/2 ensures that the squared error per example is normalized between 0 and 1: the worst possible situation is that a wrong class is predicted with certainty, in which case two "bits" are wrong (Brier did not include this factor $1/2$ in his original account; Brier 1950). For binary classification this expression can be simplified to $BS = 1/|T| \sum_{x \in T} (p(x) - b(x))^2$, where $p(x)$ is the predicted probability of a designated class (the true class, say) and $b(x)$ is 1 if the designated class is the actual one and 0 otherwise.

Suppose we want to assign the same probability vector $(p_1, \ldots, p_k)$ to all labeled instances in a given set S – for example, all training instances that get filtered into the same leaf of a decision tree. Which assignment results in the lowest Brier score? As it turns out, this is exactly the empirical class distribution in the set, which will be denoted $(\dot{p}_1, \ldots, \dot{p}_k)$. This follows from the fact that the Brier score over S can be decomposed as

$$BS(S) = \frac{1}{2}\sum_{i=1}^{k}(p_i - \dot{p}_i)^2 + \frac{1}{2}\sum_{i=1}^{k}\dot{p}_i(1 - \dot{p}_i)$$

The first term in this decomposition is known as the *calibration loss*, and the second term is called the *refinement loss*. As both terms in this decomposition are nonnegative and refinement loss is independent of p_i, the overall expression is minimized by minimizing calibration loss, which gives $p_i = \dot{p}_i$ for all i.

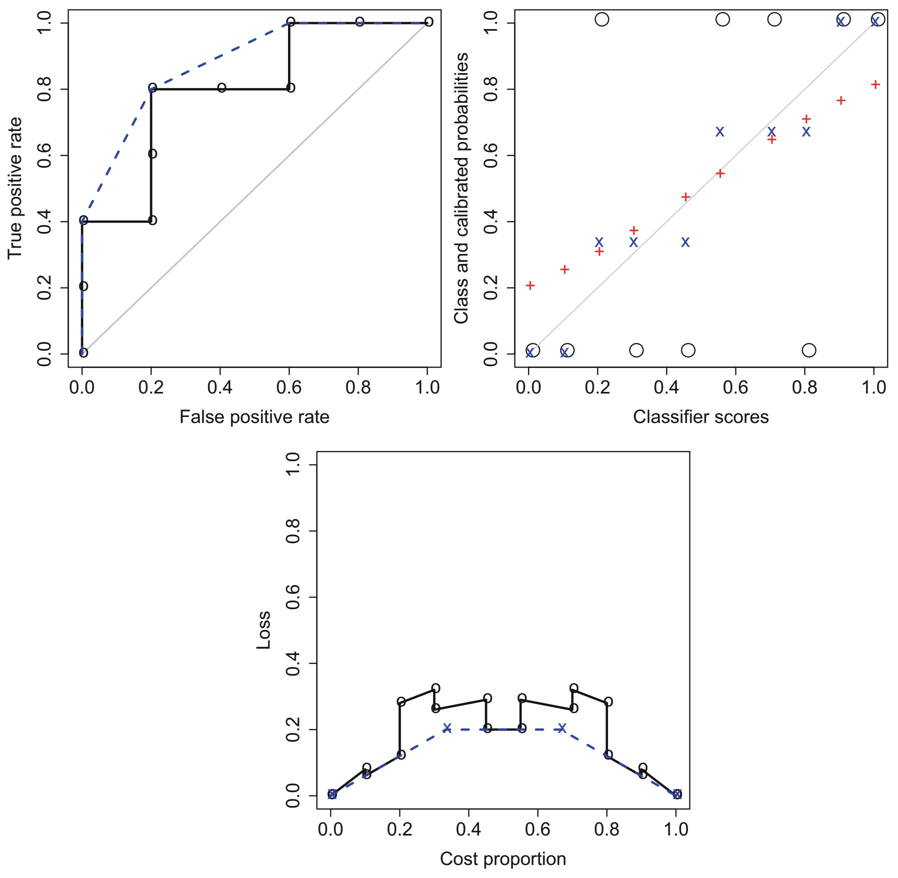
By taking a weighted average over all leaves of a decision tree, the decomposition can be applied to the Brier score over the entire data set. However, for a linear classifier which potentially assigns unique probabilities to every test instance, we need some way to group instances with similar probabilities together so that we can calculate the empirical probabilities. There are several ways of achieving this grouping, but the decomposition will in general be approximate (unless we have access to the true distributions, for which Hernández-Orallo et al. (2012) give the exact decomposition). Nevertheless, the important point to note is that Brier score is a combined measure of how well calibrated a classifier is and how well separated the scores for positives and negatives are.

The fact that Brier score is minimized if the predicted probabilities match the empirical probabilities identifies it as a so-called proper scoring rule (Gneiting and Raftery 2007). Many other proper scoring rules exist, including logarithmic loss which penalizes a probability vector with the negative logarithm of the probability assigned to the true class. These other scoring rules are amenable to similar decompositions: e.g., logarithmic loss uses Kullback-Leibler divergence between predicted and empirical probabilities to quantify calibration loss and Shannon entropy to quantify refinement loss of the empirical probabilities. Kull and Flach (2015) give an overview and also provides an underlying four-way decomposition which includes a model-independent irreducible component as well as a component that can be reduced to zero by a simple affine transformation of the scores.

Calibration Methods

In order to understand what it means for a classifier to be calibrated, it is instructive to consider its ROC curve (see ▶ ROC Analysis). A ROC curve plots true positive rate against false positive rate when the decision threshold is varied, and its slope is proportional to the empirical probability among instances receiving the same (or similar) scores from the classifier. Consider a small example in which a classifier assigns scores $(1.00, 0.90, 0.80, 0.70, 0.55, 0.45, 0.30, 0.20, 0.10, 0.0)$ to 10 test instances with true classes $(1, 1, 0, 1, 1, 0, 0, 1, 0, 0)$, which is visualized in Fig. 1. The first thing we note is that the ROC curve on the top left has "dents" or concavities where the slope – and therefore the empirical probability – increases but the scores decrease (e.g., scores $(0.80, 0.70, 0.55)$ with empirical probabilities $(0, 1, 1)$). We can "repair" these concavities by tying the scores $(2/3, 2/3, 2/3)$, thereby forming the ROC convex hull (dashed line in the top left figure) and a piecewise constant calibration map (crosses in the middle figure).

The idea of using the ROC convex hull as a calibration method is related to isotonic regression, which differs from standard least-



Classifier Calibration, Fig. 1 (*top left*) Example ROC curve (*black*, *solid line*) and convex hull (*blue*, *dashed line*) on a small data set with 10 instances. (*top right*) Uncalibrated scores against true classes (*black circles*), calibrated scores obtained with isotonic regression (*blue crosses*), and logistic calibration (*red plusses*); the effect of calibration is larger for points further away from the diagonal. (*bottom*) Cost curves obtained when thresholding at cost proportion *c* for the original, uncalibrated scores (*black*, *solid line*) and isotonically calibrated scores (*blue*, *dashed line*); the difference between the two curves represents the decrease in Brier score achievable by calibration

squares regression in that the fitted line be piecewise constant. The standard algorithm to perform isotonic regression is the *pool adjacent violators* (PAV) method, which was introduced to the machine learning community by Zadrozny and Elkan (2001). The method, which we will call isotonic calibration in this entry, was demonstrated to be equivalent to the ROC convex hull algorithm by Fawcett and Niculescu-Mizil (2007). The key idea is that the slope of a ROC curve segment represents an empirical likelihood ratio *LR* and hence

$$p = \frac{LR}{LR + a} \tag{4}$$

with $a = (1 - \pi)/\pi$ is the corresponding empirical posterior probability (where π is the proportion of positives). See ▸ ROC Analysis for further details.

Alternatively, we can obtain *LR* from a parametric model. For example, suppose that the scores are obtained from a linear model $s(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} - t$ and assume for simplicity that the weight vector $\mathbf{w}$ is unit length, then $s(\mathbf{x})$ gives the signed distance of $\mathbf{x}$ from the decision boundary in the direction given by $\mathbf{w}$. The parametric assumption is that these distances are normally distributed within each class with the same variance σ^2, from which we can derive

$$
\begin{aligned}
LR(\mathbf{x}) &= \exp(\gamma(\mathbf{w}\cdot\mathbf{x} - t')) \\
&= \exp(\gamma(s(\mathbf{x}) - (t' - t))) \\
\gamma &= \mathbf{w}\cdot(\mu^{+} - \mu^{-})/\sigma^{2} \\
t' &= \mathbf{w}\cdot(\mu^{+} + \mu^{-})/2
\end{aligned}
$$

where μ^{+} and μ^{-} are the class means. Plugging this back into Eq. (4) and assuming $a = 1$ for simplicity gives

$$
\begin{aligned}
p(\mathbf{x}) &= \frac{1}{1 + \exp(-\gamma(\mathbf{w}\cdot\mathbf{x} - t'))} \\
&= \frac{1}{1 + \exp(-\gamma(s(\mathbf{x}) - (t' - t)))}
\end{aligned}
$$

Interpreted as a mapping from s to p, this defines a logistic curve with midpoint $s = t'-t$ and slope γ at this midpoint. The location parameter $t' - t$ shifts the decision boundary such that it cuts the line connecting the two class means halfway; the shape parameter γ quantifies how well separated the two classes are. This method was popularized by John Platt as a way to get probabilities out of ▸ support vector machines (Platt 2000) and is often referred to as *Platt scaling*; we call it logistic calibration in this entry.

Figure 1 shows the results of both logistic and isotonic calibration on the running example. The isotonically calibrated classifier both has higher AUC than the uncalibrated one (0.88 rather than 0.80) – since tying the scores as suggested by the ROC convex hull leads to a better ranking – and also lower Brier score (0.1333 rather than 0.1885). The latter is visualized in the cost curves on the bottom, which plot the cost-sensitive loss for different values of the cost proportion c (Drummond and Holte 2006). Treating the uncalibrated scores as if they were calibrated and hence using c for the decision threshold as derived in Eq. (2) yields the upper, solid cost curve, the area under which is equal to the uncalibrated Brier score (Hernández-Orallo et al. 2011). Using the isotonically calibrated scores yields the lower, dashed cost curve upon which we cannot improve without changing the model.

Also shown in the top right plot are the logistically calibrated scores; as this is a monotonic transformation, it doesn't affect the ranking or the ROC curve. The model assumptions of logistic calibration (normally distributed scores per class) are not really satisfied on this small example, and hence the logistically calibrated scores only lead to a small decrease in Brier score (0.1871) with a cost curve similar to the uncalibrated one (not shown).

If we decompose the Brier score into calibration loss and refinement loss, we see that for the uncalibrated scores refinement loss is zero as no instances from different classes receive the same score, so the Brier score equals calibration loss. Conversely, we see that for the isotonically calibrated scores, the calibration loss is zero, and hence the Brier score equals the refinement loss (on the labeled data on which calibration was carried out). Hence, isotonic calibration increases the refinement loss in order to achieve a larger decrease in calibration loss. In contrast, logistic calibration only affects calibration loss.

Practically speaking, calibration methods require a separate calibration set to avoid overfitting. For experimental studies on classifier calibration methods, see Niculescu-Mizil and Caruana (2005) and Bella et al. (2013).

Future Directions

Multiclass and Multilabel Calibration

Calibration of a multiclass classifier is not a solved problem, but there are several possible strategies. As logistic calibration essentially involves fitting a univariate logistic regression model to the scores output by the classifier, a natural way to extend this to more than two classes is to use multinomial logistic regression. This is again a parametric model which assumes that scores are normally distributed within each class.

A simple method that is recommended by Zadrozny and Elkan (2002) is to perform a logistic or isotonic calibration for each class separately, treating all other classes as the negative class (one-versus-rest). Since the resulting probabilities don't necessarily add up to one, they can be renormalized by dividing them by their sum.

Alternatively, one might consider to calibrate each class against each other class (one-versus-one). This results in more probabilities than there are classes, and hence methods to produce a multinomial probability vector are more involved. Hastie and Tibshirani (1998) proposed a solution called coupling, and Zadrozny (2001) generalized it to other code matrices (see ▶ Error Correcting Output Codes). Kong and Dietterich (1997) proposed an alternative method, also based on ECOC.

Multilabel classifiers differ from multiclass classifiers in that several labels may apply simultaneously. In the presence of sparse labels, calibration is particularly important, but it needs to be considered in the context of how multilabel classification performance will be evaluated. For example, one popular evaluation metric is Hamming loss which calculates the proportion of labels that are mispredicted – this effectively puts all labels in the same bag, and hence there is no point in separately calibrating the scores for each label. Other evaluation metrics are calculated label-wise or even instance-wise. Further research is needed to identify the right calibration methods for these cases.

Calibrating for Different Losses

Throughout this entry we have made an implicit assumption that our goal is to maximize accuracy: the proportion of correctly classified instances (true positives and true negatives). This justifies the additive cost model of Eq. (1) which led to the model-independent thresholds summarized in Eqs. (2) and (3). However, there is a range of recent results demonstrating that this threshold is suboptimal for other performance metrics. For example, Zhao et al. (2013) proved that if our goal is to maximize the F-score (the harmonic mean of precision and recall), then the optimal threshold θ^* on the true posterior is half the F-score obtained at that threshold – it follows that the optimal threshold for F-score is less than or equal to the optimal threshold for accuracy, with equality obtained only for perfect classifiers. Koyejo et al. (2014) extends the analysis to a family of performance metrics including accuracy and the F_β-score (a weighted harmonic mean of precision and recall, with $\beta = 1$ yielding the F-score). Specifically, if F_β^* denotes the optimal F_β-score achievable by a model, then $\theta^* = F_\beta^*/(1+\beta^2)$.

Instead of investigating how to adapt the decision rule on the class posterior to account for a different performance metric – which is necessarily classifier dependent – Flach and Kull (2015) suggest for the classifier to output a different score specifically adapted for the F_β metric. Let p be the calibrated posterior probability for a given instance; let B be the value of β for which the instance is on the F_β decision boundary (i.e., the F_β-score would be the same regardless of whether the instance is predicted to be positive or negative); and let F_B^* be the optimal F_β-score achievable by the model for that value of $\beta = B$; then the model outputs the score $p/F_B^* = 1/(1+B^2)$. All quantities involved can be precomputed from the ROC convex hull. This naturally leads to the idea of a classifier outputting *multiple* calibrated scores: calibrated estimates of the posterior probability for optimizing accuracy, adjustments of the posterior as just described for optimizing F-scores, and possibly others. The latter can still be seen as calibrated scores if we adopt a broader view of calibration: *a well-calibrated classifier calculates the cost parameters under which the expected cost for the instance under consideration is the same regardless of the predicted class.*

Cross-References

▶ Classification
▶ Class Imbalance Problem
▶ Cost-Sensitive Learning
▶ Logistic Regression
▶ Posterior Probability
▶ ROC Analysis

Recommended Reading

Bella A, Ferri C, Hernández-Orallo J, Ramírez-Quintana MJ (2013) On the effect of calibration in classifier combination. Appl Intell 38(4):566–585

Brier G (1950) Verification of forecasts expressed in terms of probabilities. Mon Weather Rev 78:1–3
Drummond C, Holte R (2006) Cost curves: an improved method for visualizing classifier performance. Mach Learn 65(1):95–130
Elkan C (2001) The foundations of cost-sensitive learning. In: Proceedings of 17th international joint conference on artificial intelligence (IJCAI'01). Morgan Kaufmann, pp 973–978
Fawcett T, Niculescu-Mizil A (2007) PAV and the ROC convex hull. Mach Learn 68(1):97–106
Ferri C, Flach P, Hernández-Orallo J (2003) Improving the AUC of probabilistic estimation trees. In: 14th European conference on machine learning (ECML'03). Springer, pp 121–132
Flach P, Kull M (2015) Precision-recall-gain curves: PR analysis done right. In: Advances in neural information processing systems (NIPS'15), pp 838–846
Gneiting T, Raftery AE (2007) Strictly proper scoring rules, prediction, and estimation. J Am Stat Assoc 102(477):359–378
Hastie T, Tibshirani R (1998) Classification by pairwise coupling. Ann Stat 26(2):451–471
Hernández-Orallo J, Flach P, Ferri C (2011) Brier curves: a new cost-based visualisation of classifier performance. In: Proceedings 28th international conference on machine learning (ICML'11), pp 585–592
Hernández-Orallo J, Flach P, Ferri C (2012) A unified view of performance metrics: translating threshold choice into expected classification loss. J Mach Learn Res 13(1):2813–2869
Kong EB, Dietterich T (1997) Probability estimation via error-correcting output coding. In: International conference on artificial intelligence and soft computing
Koyejo OO, Natarajan N, Ravikumar PK, Dhillon IS (2014) Consistent binary classification with generalized performance metrics. In: Advances in neural information processing systems (NIPS'14), pp 2744–2752
Kull M, Flach P (2015) Novel decompositions of proper scoring rules for classification: score adjustment as precursor to calibration. In: Machine learning and knowledge discovery in databases (ECML-PKDD'15). Springer, pp 68–85
Niculescu-Mizil A, Caruana R (2005) Predicting good probabilities with supervised learning. In: Proceedings of 22nd international conference on machine learning (ICML'05), pp 625–632
Platt J (2000) Probabilities for SV machines. In: Smola A, Bartlett P, Schölkopf B, Schuurmans D (eds) Advances in large margin classifiers. MIT Press, Cambridge, pp 61–74
Provost F, Domingos P (2003) Tree induction for probability-based ranking. Mach Learn 52(3):199–215
Zadrozny B (2001) Reducing multiclass to binary by coupling probability estimates. In: Advances in neural information processing systems (NIPS'01), pp 1041–1048
Zadrozny B, Elkan C (2001) Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In: Proceedings of 18th international conference on machine learning (ICML'01), pp 609–616
Zadrozny B, Elkan C (2002) Transforming classifier scores into accurate multiclass probability estimates. In: Proceedings of 8th international conference on knowledge discovery and data mining (KDD'02). ACM, pp 694–699
Zhao M-J, Edakunni N, Pocock A, Brown G (2013) Beyond Fano's inequality: bounds on the optimal F-score, BER, and cost-sensitive risk and their implications. J Mach Learn Res 14(1):1033–1090

Classifier Systems

Pier Luca Lanzi
Politecnico di Milano, Milano, Italy

Synonyms

Genetics-based machine learning; Learning classifier systems

Definition

Classifier systems are rule-based systems that combine ▶ temporal difference learning or ▶ supervised learning with a genetic algorithm to solve classification and ▶ reinforcement learning problems. Classifier systems come in two flavors: Michigan classifier systems, which are designed for online learning, but can also tackle offline problems; and Pittsburgh classifier systems, which can only be applied to offline learning.

In Michigan classifier systems (Holland 1976), learning is viewed as an online adaptation process to an unknown environment that represents the problem and provides feedback in terms of a numerical reward. Michigan classifier systems maintain a single candidate solution consisting of a set of rules, or a population of classifiers. Michigan systems apply (1) temporal difference learning to distribute the incoming reward to the classifiers that are accountable for it; and (2) a genetic algorithm to select, recombine, and mutate individual classifiers so as to improve their contribution to the current solution.

In contrast, in Pittsburgh classifier systems (Smith 1980), learning is viewed as an offline optimization process in which a genetic algorithm alone is applied to search for the best solution to a given problem. In addition, Pittsburgh classifier systems maintain not one, but a set of candidate solutions. While in the Michigan classifier system each individual classifier represents a part of the overall solution, in the Pittsburgh system each individual is a complete candidate solution (itself consisting of a set of classifiers). The fitness of each Pittsburgh individual is computed offline by testing it on a representative sample of problem instances. The individuals compete among themselves through selection, while crossover and mutation recombine solutions to search for better solutions.

Motivation and Background

Machine learning is usually viewed as a search process in which a solution space is explored until an appropriate solution to the target problem is found (Mitchell 1982) (see ▶ Supervised Learning). Machine learning methods are characterized by the way they represent solutions (e.g., using ▶ decision trees, rules), by the way they evaluate solutions (e.g., classification accuracy, information gain) and by the way they explore the solution space (e.g., using a ▶ general-to-specific strategy or a specific-to-general strategy).

Classifier systems are methods of *genetics-based* machine learning introduced by Holland, the father of ▶ genetic algorithms. They made their first appearance in Holland (1976) where the first diagram of a classifier system, labeled "cognitive system," was shown. Subsequently, they were described in detail in the paper "Cognitive Systems based on Adaptive Algorithms" (Holland and Reitman 1978). Classifier systems are characterized by a rule-based representation of solutions and a genetics-based exploration of the solution space. While other ▶ rule learning methods, such as CN2 (Clark and Niblett 1989) and FOIL (Quinlan and Cameron-Jones 1995), generate one rule at a time following a sequential covering strategy (see ▶ Covering Algorithm), classifier systems work on one or more solutions at once, and they explore the solution space by applying the principles of natural selection and genetics.

In classifier systems (Holland 1976; Holland and Reitman 1978; Wilson 1995), machine learning is modeled as an online adaptation process to an unknown *environment*, which provides feedback in terms of a numerical reward. A classifier system perceives the environment through its detectors and, based on its sensations, it selects an action to be performed in the environment through its effectors. Depending on the efficacy of its actions, the environment may eventually reward the system. A classifier system learns by trying to maximize the amount of reward it receives from the environment. To pursue such a goal, it maintains a set (a *population*) of condition-action-prediction rules, called *classifiers*, which represents the current solution. Each classifier's condition identifies some part of the problem domain; the classifier's action represents a decision on the subproblem identified by its condition; and the classifier's prediction, or strength, estimates the value of the action in terms of future rewards on that subproblem. Two separate components, credit assignment and rule discovery, act on the population with different goals. ▶ Credit assignment, implemented either by methods of temporal difference or supervised learning, exploits the incoming reward to estimate the action values in each subproblem so as to identify the best classifiers in the population. At the same time, rule discovery, usually implemented by a genetic algorithm, selects, recombines, and mutates the classifiers in the population to improve the current solution.

Classifier systems were initially conceived as modeling tools. Given a real system with unknown underlying dynamics, for instance a financial market, a classifier system would be used to generate a behavior that matched the real system. The evolved rules would provide a plausible, human readable model of the unknown system – a way to look inside the box. Subsequently, with the developments in the area of machine learning and the rise of reinforcement learning, classifier systems have been more and more often studied and presented as alternatives to other machine learning methods. Wilson's XCS (1995), the most successful classifier system to date, has proven to be both a valid alternative to other reinforcement learning approaches and an effective approach to classification and data mining (Bull 2004; Bull and Kovacs 2005; Lanzi et al. 2000).

Kenneth de Jong and his students (de Jong 1988; Smith 1980, 1983) took a different perspective on genetics-based machine learning and modeled learning as an *optimization* process rather than an *adaptation* process as done in Holland (1976). In this case, the solution space is explored by applying a genetic algorithm to a population of individuals each representing a *complete* candidate solution – that is, a set of rules (or a production system, de Jong 1988; Smith 1980. At each cycle, a critic is applied to each individual (to each set of rules) to obtain a performance measure that is then used by the genetic algorithm to guide the exploration of the solution space. The individuals in the population compete among themselves through selection, while crossover and mutation recombine solutions to search for better ones.

The approaches of Holland (Holland 1976; Holland and Reitman 1978) and de Jong (de Jong 1988; Smith 1980, 1983) have been extended and improved in several ways (see Lanzi et al. (2000) for a review). The models of classifier systems that are inspired by the work of Holland (1976) at the University of Michigan are usually called Michigan classifier systems; the ones that are inspired by Smith (1980, 1983) and de Jong (1988) at the University of Pittsburgh are usually termed Pittsburgh classifier systems – or briefly, Pitt classifier systems.

Pittsburgh classifier systems separate the evaluation of candidate solutions, performed by an external critic, from the genetic search. As they evaluate candidate solutions as a whole, Pittsburgh classifier systems can easily identify and emphasize sequentially cooperating classifiers, which is particularly helpful in problems involving partial observability. In contrast, in Michigan classifier systems the credit assignment is focused, due to identification of the actual classifiers that produce the reward, so learning is *much* faster but sequentially cooperating classifiers are more difficult to spot. As Pittsburgh classifier systems apply the genetic algorithm to a set of solutions, they only work offline, whereas Michigan classifier systems work online, although they can also tackle offline problems. Finally, the design of Pittsburgh classifier systems involves decisions as to how an entire solution should be represented and how solutions should be recombined – a task which can be daunting. In contrast, the design of Michigan classifier systems involves simpler decisions about how a rule should be represented and how two rules should be recombined. Accordingly, while the representation of solutions and its related issues play a key role in Pittsburgh models, Michigan models easily work with several types of representations (Lanzi 2001; Lanzi and Perrucci 1999; Mellor 2005).

Structure of the Learning System

Michigan and Pittsburgh classifier systems were both inspired by the work of Holland on the broadcast language (Holland 1975). However, their structures reflect two different ways to model machine learning: as an adaptation process in the case of Michigan classifier systems; and as an optimization problem, in the case of Pittsburgh classifier systems. Thus, the two models, originating from the same idea (Holland's broadcast language), have radically different structures.

Michigan Classifier Systems

Holland's classifier systems define a general paradigm for genetics-based machine learning. The description in Holland and Reitman (1978) provides a list of principles for online learning through adaptation. Over the years, such principles have guided researchers who developed several models of Michigan classifier systems (Butz 2002; Wilson 1994, 1995, 2002) and applied them to a large variety of domains

(Bull 2004; Lanzi and Riolo 2003; Lanzi et al. 2000). These models extended and improved Holland's original ideas, but kept all the ingredients of the original recipe: a population of classifiers, which represents the current system knowledge; a performance component, which is responsible for the short-term behavior of the system; a credit assignment (or reinforcement) component, which distributes the incoming reward among the classifiers; and a rule discovery component, which applies a genetic algorithm to the classifiers to improve the current knowledge.

Knowledge Representation

In Michigan classifier systems, knowledge is represented by a population of classifiers. Each classifier is usually defined by four main parameters: the *condition*, which identifies some part of the problem domain; the *action*, which represents a decision on the subproblem identified by its condition; the *prediction* or strength, which estimates the amount of reward that the system will receive if its action is performed; and finally, the *fitness*, which estimates how good the classifier is in terms of problem solution.

The knowledge representation of Michigan classifier systems is extremely flexible. Each one of the four classifier components can be tailored to fit the need of a particular application, without modifying the main structure of the system. In problems involving binary inputs, classifier conditions can be simply represented using strings defined over the alphabet {0, 1, #}, as done in Holland and Reitman (1978), Goldberg (1989), and Wilson (1995). In problems involving real inputs, conditions can be represented as disjunctions of intervals, similar to the ones produced by other rule learning methods (Clark and Niblett 1989). Conditions can also be represented as general-purpose symbolic expressions (Lanzi 2001; Lanzi and Perrucci 1999) or first-order logic expressions (Mellor 2005). Classifier actions are typically encoded by a set of symbols (either binary strings or simple labels), but continuous real-valued actions are also available (Wilson 2007). Classifier prediction (or strength) is usually encoded by a parameter (Goldberg 1989; Holland and Reitman 1978; Wilson 1995). However, classifier prediction can also be computed using a parameterized function (Wilson 2002), which results in solutions represented as an ensemble of local approximators – similar to the ones produced in generalized reinforcement learning (Sutton and Barto 1998).

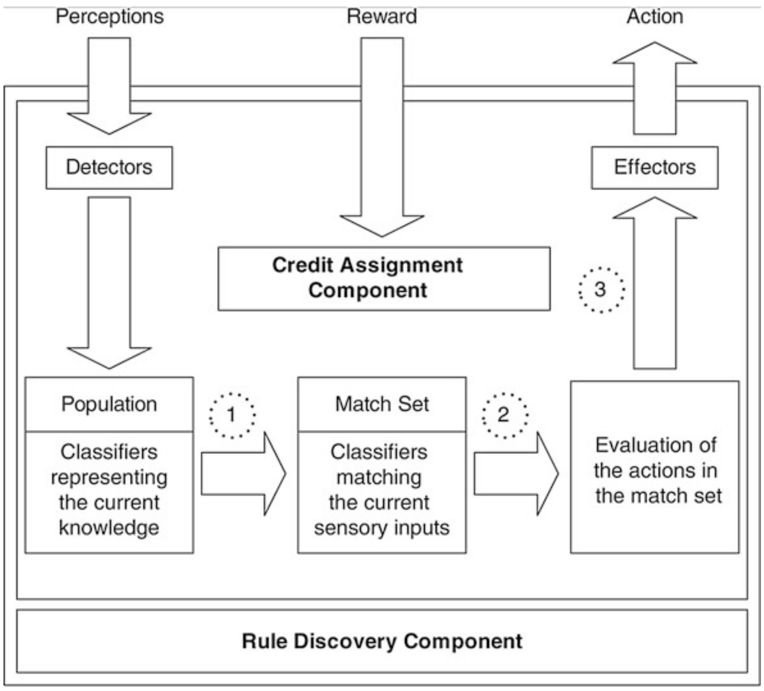
Performance Component

A simplified structure of Michigan classifier systems is shown in Fig. 1. We refer the reader to Goldberg (1989) and Holland and Reitman (1978) for a detailed description of the original model and to Butz (2002) and Wilson (1994, 1995, 2001) for descriptions of recent classifier system models.

A classifier system learns through trial and error interactions with an unknown environment. The system and the environment interact continually. At each time step, the classifier system perceives the environment through its detectors; it builds a *match set* containing all the classifiers in the population whose condition matches the current sensory input. The match set typically contains classifiers that advocate contrasting actions; accordingly, the classifier system evaluates each action in the match set, and selects an action to be performed balancing exploration and exploitation. The selected action is sent to the effectors to be executed in the environment; depending on the effect that the action has in the environment, the system receives a scalar reward.

Credit Assignment

The *credit assignmentcomponent* (also called reinforcement component, Wilson 1995) distributes the incoming reward to the classifiers that are accountable for it. In Holland and Reitman (1978), credit assignment is implemented by Holland's bucket brigade algorithm (Holland 1986), which was partially inspired by the credit allocation mechanism used by Samuel in his pioneering work on learning checkers-playing programs (Samuel 1959).


Perceptions
Reward
Action
Detectors
Effectors
Credit Assignment Component
3
Population
Classifiers representing the current knowledge
1
Match Set
Classifiers matching the current sensory inputs
2
Evaluation of the actions in the match set
Rule Discovery Component

Classifier Systems, Fig. 1 Simplified structure of a Michigan classifier system. The system perceives the environment through its detectors and (1) it builds the match set containing the classifiers in the population that match the current sensory inputs; then (2) all the actions in the match set are evaluated, and (3) an action is selected to be performed in the environment through the effectors

In the early years, classifier systems and the bucket brigade algorithm were confined to the evolutionary computation community. The rise of reinforcement learning increased the connection between classifier systems and temporal difference learning (Sutton 1988; Sutton and Barto 1998): in particular, Sutton (1988) showed that the bucket brigade algorithm is a kind of temporal difference learning, and similar connections were also made in Watkins (1989) and Dorigo and Bersini (1994). Later, the connection between classifier systems and reinforcement learning became tighter with the introduction of Wilson's XCS (1995), in which credit assignment is implemented by a modification of Watkins Q-learning (Watkins 1989). As a consequence, in recent years, classifier systems are often presented as methods of reinforcement learning with genetics-based generalization (Bull and Kovacs 2005).

Rule Discovery Component

The *rule discovery component* is usually implemented by a genetic algorithm that selects classifiers in the population with probability proportional to their fitness; it copies the selected classifiers and applies genetic operators (usually crossover and mutation) to the offspring classifiers; the new classifiers are inserted in the population, while other classifiers are deleted to keep the population size constant.

Classifiers selection plays a central role in rule discovery. Classifier selection depends on the definition of classifier fitness and on the subset of classifiers considered during the selection process. In Holland and Reitman (1978), classifier fitness coincides with classifier prediction, while selection is applied to all the classifiers in the population. This approach results in a pressure toward classifiers predicting high returns, but typically tends to produce overly general solutions. To avoid such solutions, Wilson (1995) introduced the XCS classifier system in which accuracy-based fitness is coupled with a niched genetic algorithm. This approach results in a pressure toward accurate maximally general classifiers, and has made XCS the most successful classifier system to date.

Pittsburgh Classifier Systems

The idea underlying the development of Pittsburgh classifier systems was to show that interesting behaviors could be evolved using a simpler model than the one proposed by Holland with Michigan classifier systems (Holland 1976; Holland and Reitman 1978).

In Pittsburgh classifier systems, each individual is a set of rules that encodes an entire candidate solution; each rule has a fixed length, but each rule set (each individual) usually contains a variable number of rules. The genetic operators, crossover and mutation, are tailored to the rule-based, variable-length representation. The individuals in the population compete among themselves, following the selection-recombination-mutation cycle that is typical of genetic algorithms (Goldberg 1989; Holland 1975). While in Michigan classifier systems individuals in the population (the single rules) cooperate, in Pittsburgh classifier systems there is no cooperation among individuals (the rule sets), so that the genetic algorithm operation is simpler for Pittsburgh models. However, as Pittsburgh classifier systems explore a much larger search space, they usually require more computational resources than Michigan classifier systems.

The pseudo-code of a Pittsburgh classifier system is shown in Fig. 2. At first, the individuals in the population are randomly initialized (line 2). At time t, the individuals are evaluated by an external critic, which returns a performance measure that the genetic algorithm exploits to compute the fitness of individuals (lines 3 and 10). Following this, selection (line 6), recombination, and mutation (line 7) are applied to the individuals in the population – as done in a typical genetic algorithm. The process stops when a termination criterion is met (line 4), usually when an appropriate solution is found.

The design of Pittsburgh classifier systems follows the typical steps of genetic algorithm design, which means deciding how a rule set should be represented, what genetic operators should be applied, and how the fitness of a set of rules should be calculated. In addition, Pittsburgh classifier systems need to address the *bloat* phenomenon (Tackett 1994) that arises with any variable-sized representation, like the rule sets evolved by Pittsburgh classifier systems. Bloat can be defined as the growth of individuals without an actual fitness improvement. In Pittsburgh classifier systems, bloat increases the size of candidate solutions by adding useless rules to individuals, and it is typically limited by introducing a parsimony pressure that discourages large rule sets (Bassett and de Jong 2000). Alternatively, Pittsburgh classifier systems can be combined with multi-objective optimization, so as to separate the maximization of the rule set performance and the minimization of the rule set size.

Examples of Pittsburgh classifier systems include SAMUEL (Grefenstette et al. 1990), the Genetic Algorithm Batch-Incremental Concept Learner (GABIL) (de Jong and Spears 1991), GIL Janikow (1993), GALE (Llorá 2002), and GAssist (Bacardit 2004).

Applications

Classifier systems have been applied to a large variety of domains, including computational

Classifier Systems, Fig. 2 Pseudo-code of a Pittsburgh classifier system

```
1.   t := 0
2.   Initialize the population P(t)
3.   Evaluate the rules sets in P(t)
4.   **While** the termination condition is not satisfied
5.   **Begin**
6.      Select the rule sets in P(t) and generate Ps(t)
7.      Recombine and mutate the rule sets in Ps(t)
8.      P(t+1) := Ps(t)
9.      t := t+1
10.     Evaluate the rules sets in P(t)
11.  **End**
```

economics (e.g., Arthur et al. 1996), autonomous robotics (e.g., Dorigo and Colombetti 1998), classification (e.g., Barry et al. 2004), fighter aircraft maneuvering (Bull 2004; Smith et al. 2000), and many others. Reviews of classifier system applications are available in Lanzi et al. (2000); Lanzi and Riolo (2003), and Bull (2004).

Programs and Data

The major sources of information about classifier systems are the LCSWeb maintained by Alwyn Barry, which can be reached through, and www.learning-classifier-systems.org maintained by Xavier Llorà.

Several implementations of classifier systems are freely available online. The first standard implementation of Holland's classifier system in Pascal was described in Goldberg (1989), and it is available at http://www.illigal.org/; a C version of the same implementation, developed by Robert E. Smith, is available at http://www.etsimo.uniovi.es/ftp/pub/EC/CFS/src/. Another implementation of an extension of Holland's classifier system in C by Rick L. Riolo is available at http://www.cscs.umich.edu/Software/Contents.html. Implementations of Wilson's XCS 1995 are distributed by Alwyn Barry at the LCSWeb, by Martin V. Butz (at www.illigal.org), and by Pier Luca Lanzi (at xcslib.sf.net). Among the implementations of Pittsburgh classifier systems, the Samuel system is available from Alan C. Schultz at http://www.nrl.navy.mil/; Xavier Llorà distributes GALE (Genetic and Artificial Life Environment) a fine-grained parallel genetic algorithm for data mining at www.illigal.org/xllora.

Cross-References

- ▶ Credit Assignment
- ▶ Genetic and Evolutionary Algorithms
- ▶ Reinforcement Learning
- ▶ Rule Learning

Recommended Reading

Arthur BW, Holland JH, LeBaron B, Palmer R, Talyer P (1996) Asset pricing under endogenous expectations in an artificial stock market. Technical report, Santa Fe Institute

Bacardit i Peñarroya J (2004) Pittsburgh genetic-based machine learning in the data mining era: representations, generalization, and run-time. PhD thesis, Computer Science Department, Enginyeria i Arquitectura La Salle Universitat Ramon Llull, Barcelona

Barry AM, Holmes J, Llora X (2004) Data mining using learning classifier systems. In: Bull L (ed) Applications of learning classifier systems, studies in fuzziness and soft computing, vol 150. Springer, Pagg, pp 15–67

Bassett JK, de Jong KA (2000) Evolving behaviors for cooperating agents. In: Proceedings of the twelfth international symposium on methodologies for intelligent systems. LNAI, vol 1932. Springer, Berlin

Booker LB (1989) Triggered rule discovery in classifier systems. In: Schaffer JD (ed) Proceedings of the 3rd international conference on genetic algorithms (ICGA89). Morgan Kaufmann, San Francisco

Bull L (ed) (2004) Applications of learning classifier systems, studies in fuzziness and soft computing, vol 150. Springer, Berlin. ISBN 978-3-540-21109-9

Bull L, Kovacs T (eds) (2005) Foundations of learning classifier systems, studies in fuzziness and soft computing, vol 183. Springer, Berlin. ISBN 978-3-540-25073-9

Butz MV (2002) Anticipatory learning classifier systems. Genetic algorithms and evolutionary computation. Kluwer, Boston Academic Publishers.

Clark P, Niblett T (1989) The CN2 induction algorithm. Mach Learn 3(4):261–283

de Jong K (1988) Learning with genetic algorithms: an overview. Mach Learn 3(2–3):121–138

de Jong KA, Spears WM (1991) Learning concept classification rules using genetic algorithms. In: Proceedings of the international joint conference on artificial intelligence. Morgan Kaufmann, San Francisco, pp 651–656

Dorigo M, Bersini H (1994) A comparison of Q-learning and classifier systems. In: Cliff D, Husbands P, Meyer J-A, Wilson SW (eds) From animals to animats 3: proceedings of the third international conference on simulation of adaptive behavior. MIT Press, Cambridge, pp 248–255

Dorigo M, Colombetti M (1998) Robot shaping: an experiment in behavior engineering. MIT Press/Bradford Books, Cambridge

Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading

Grefenstette JJ, Ramsey CL, Schultz A (1990) Learning sequential decision rules using simulation models and competition. Mach Learn 5(4):355–381

Holland J (1986) Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning, an artificial intelligence approach, vol II, Chap. 20. Morgan Kaufmann, San Francisco, pp 593–623
Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor (Reprinted by the MIT Press in 1992)
Holland JH (1976) Adaptation. Progress in theoretical biology 4:263–293
Holland JH, Reitman JS (1978) Cognitive systems based on adaptive algorithms. In: Waterman DA, Hayes-Roth F (eds) Pattern-directed inference systems. Academic Press, New York (Reprinted from Evolutionary computation. The fossil record. Fogel DB (ed.) IEEE Press (1998))
Janikow CZ (1993) A knowledge-intensive genetic algorithm for supervised learning. Mach Learn 13(2–3):189–228
Lanzi PL (2001) Mining interesting knowledge from data with the XCS classifier system. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt H-M, Gen M et al (eds) Proceedings of the genetic and evolutionary computation conference (GECCO-2001). Morgan Kaufmann, San Francisco, pp 958–965
Lanzi PL (2005) Learning classifier systems: a reinforcement learning perspective. In: Bull L, Kovacs T (eds) Foundations of learning classifier systems, studies in fuzziness and soft computing. Springer, Berlin, pp 267–284
Lanzi PL, Perrucci A (1999) Extending the representation of classifier conditions part II: from messy coding to S-expressions. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) Proceedings of the genetic and evolutionary computation conference (GECCO 99). Morgan Kaufmann, Orlando, pp 345–352
Lanzi PL, Riolo RL (2003) Recent trends in learning classifier systems research. In: Ghosh A, Tsutsui S (eds) Advances in evolutionary computing: theory and applications. Springer, Berlin, pp 955–988
Lanzi PL, Stolzmann W, Wilson SW (eds) (2000) Learning classifier systems: from foundations to applications. Lecture notes in computer science, vol 1813. Springer, Berlin
Llorá X (2002) Genetics-based machine learning using fine-grained parallelism for data mining. PhD thesis, Enginyeria i Arquitectura La Salle, Ramon Llull University, Barcelona
Mellor D (2005) A first order logic classifier system. In: Beyer H (ed) Proceedings of the 2005 conference on genetic and evolutionary computation (GECCO '05). ACM Press, New York, pp 1819–1826
Quinlan JR, Cameron-Jones RM (1995) Induction of logic programs: FOIL and related systems. New Gener Comput 13(3–4):287–312
Samuel AL (1959) Some studies in machine learning using the game of checkers. In: Feigenbaum, Feldman J (eds) Computers and thought. McGraw-Hill, New York
Smith RE, Dike BA, Niehra RK, Ravichandran B, El-Fallah A (2000) Classifier systems in combat: two-sided learning of maneuvers for advanced fighter aircraft. Comput Methods Appl Mech Eng 186(2–4):421–437
Smith SF (1980) A learning system based on genetic adaptive algorithms. Doctoral dissertation, Department of Computer Science, University of Pittsburgh
Smith SF (1983) Flexible learning of problem solving heuristics through adaptive search. In: Proceedings of the eighth international joint conference on artificial intelligence. Morgan Kaufmann, Los Altos, pp 421–425
Sutton RS (1988) Learning to predict by the methods of temporal differences. Mach Learn 3:9–44
Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT Press, Cambridge
Tackett WA (1994) Recombination, selection, and the genetic construction of computer programs. Unpublished doctoral dissertation, University of Southern California
Watkins C (1989) Learning from delayed rewards. PhD thesis, King's College
Wilson SW (1995) Classifier fitness based on accuracy. Evol Comput 3(2):149–175
Wilson SW (2002) Classifiers that approximate functions. Natl Comput 1(2–3):211–234
Wilson SW (2007). "Three architectures for continuous action" learning classifier systems. International workshops, IWLCS 2003–2005, revised selected papers. In: Kovacs T, Llorà X, Takadama K, Lanzi PL, Stolzmann W, Wilson SW (eds) Lecture notes in artificial intelligence, vol 4399. Springer, Berlin, pp 239–257

Clause

A *clause* is a logical rule in a ▶ logic program. Formally, a clause is a disjunction of (possibly negated) literals, such as

$$\begin{aligned} &grandfather(x,y) \vee \neg father(x,z) \\ &\qquad \vee \neg parent(z,y) \end{aligned}$$

In the logic programming language ▶ Prolog this clause is written as

```
grandfather(X,Y) :- father(X,Z),
                    parent(Z,Y).
```

The part to the left of :- ("if") is the *head* of the clause, and the right part is its *body*. Informally, the clause asserts the truth of the head given the truth of the body. A clause with exactly one literal in the head is called a *Horn clause* or *definite clause*; logic programs mostly consist of definite clauses. A clause without a body is also called a *fact*; a clause without a head is also called a *denial*, or a *query* in a proof by refutation. The clause without head or body is called the *empty clause*: it signifies inconsistency or falsehood and is denoted □. Given a set of clauses, the *resolution* inference rule can be used to deduce logical consequences and answer queries (see ▶ First-Order Logic).

In machine learning, clauses can be used to express classification rules for structured individuals. For example, the following definite clause classifies a molecular compound as carcinogenic if it contains a hydrogen atom with charge above a certain threshold.

```
carcinogenic(M) :- atom(M,A1),
                   element(A1,h),
                   charge(A1,C1),
                   geq(C1,0.168).
```

Cross-References

▶ First-Order Logic
▶ Inductive Logic Programming
▶ Learning from Structured Data
▶ Logic Program
▶ Prolog

Clause Learning

In ▶ speedup learning, clause learning is a ▶ deductive learning technique used for the purpose of ▶ intelligent backtracking in satisfiability solvers. The approach analyzes failures at backtracking points and derives clauses that must be satisfied by the solution. The clauses are added to the set of clauses from the original satisfiability problem and serve to prune new search nodes that violate them.

Click-Through Rate (CTR)

CTR measures the success of a ranking of search results, or advertisement placing. Given the number of *impressions*, the number of times a web result or ad has been displayed, and the number of *clicks*, the number of users who clicked on the result/advertisement, CTR is the number of clicks divided by the number of impressions.

Clonal Selection

The clonal selection theory (CST) is the theory used to explain the basic response of the adaptive immune system to an antigenic stimulus. It establishes the idea that only those cells capable of recognizing an antigenic stimulus will proliferate, thus being selected against those that do not. Clonal selection operates on both T-cells and B-cells. When antibodies on a B-cell bind with an antigen, the B-cell becomes activated and begins to proliferate. New B-cell clones are produced that are an exact copy of the parent B-cell, but then they undergo somatic hypermutation and produce antibodies that are specific to the invading antigen. The B-cells, in addition to proliferating or differentiating into *plasma cells*, can differentiate into long-lived B *memory cells*. Plasma cells produce large amounts of *antibody* which will attach themselves to the antigen and act as a type of *tag* for T-cells to pick up on and remove from the system. This whole process is known as *affinity maturation*. This process forms the basis of many artificial immune system algorithms such as AIRS and aiNET.

Closest Point

▶ Nearest Neighbor

Cluster Editing

The Cluster Editing problem is almost equivalent to Correlation Clustering on complete instances. The idea is to obtain a graph that consists only of cliques. Although Cluster Deletion requires us to delete the smallest number of edges to obtain such a graph, in Cluster Editing we are permitted to add as well as remove edges. The final variant is Cluster Completion in which edges can only be added: each of these problems can be restricted to building a specified number of cliques.

Cluster Ensembles

Cluster ensembles are an unsupervised ▸ ensemble learning method. The principle is to create multiple different clusterings of a dataset, possibly using different algorithms, then aggregate the opinions of the different clusterings into an ensemble result. The final ensemble clustering should be in theory more reliable than the individual clusterings.

Cluster Initialization

▸ *K*-Means Clustering

Cluster Optimization

▸ Evolutionary Clustering

Clustering

Clustering is a type of ▸ unsupervised learning in which the goal is to partition a set of ▸ examples into groups called clusters. Intuitively, the examples within a cluster are more similar to each other than to examples from other clusters. In order to measure the similarity between examples, clustering algorithms use various distortion or ▸ distance measures. There are two major types clustering approaches: generative and discriminative. The former assumes a parametric form of the data and tries to find the model parameters that maximize the probability that the data was generated by the chosen model. The latter represents graph-theoretic approaches that compute a similarity matrix defined over the input data.

Cross-References

▸ Categorical Data Clustering
▸ Cluster Editing
▸ Cluster Ensembles
▸ Clustering
▸ Clustering from Data Streams
▸ Consensus Clustering
▸ Constrained Clustering
▸ Correlation Clustering
▸ Cross-Language Document Categorization
▸ Density-Based Clustering
▸ Dirichlet Process
▸ Evolutionary Clustering
▸ Graph Clustering
▸ *K*-Means Clustering
▸ *K*-Mediods Clustering
▸ Model-Based Clustering
▸ Partitional Clustering
▸ Projective Clustering
▸ Sublinear Clustering

Clustering Aggregation

▸ Consensus Clustering

Clustering Ensembles

▸ Consensus Clustering

Clustering from Data Streams

João Gama
University of Porto, Porto, Portugal

Abstract

Clustering is one of the most popular data mining techniques. In this article, we review the

relevant methods and algorithms for designing cluster algorithms under the data streams computational model, and discuss research directions in tracking evolving clusters.

Definition

Clustering is the process of grouping objects into different groups, such that the common properties of data in each subset are high and between different subsets are low. The data stream clustering problem is defined as *to maintain a continuously consistent good clustering of the sequence observed so far, using a small amount of memory and time*. The issues are imposed by the continuous arriving data points and the need to analyze them in real time. These characteristics require incremental clustering, maintaining cluster structures that evolve over time. Moreover, the data stream may evolve over time, and new clusters might appear, other disappears, reflecting the dynamics of the stream.

Main Techniques

Clustering data streams requires a process able to continuously cluster objects within memory and time restrictions (Gama 2010). Following Silva et al. (2013), algorithms for clustering data streams should ideally fulfill the following requirements:

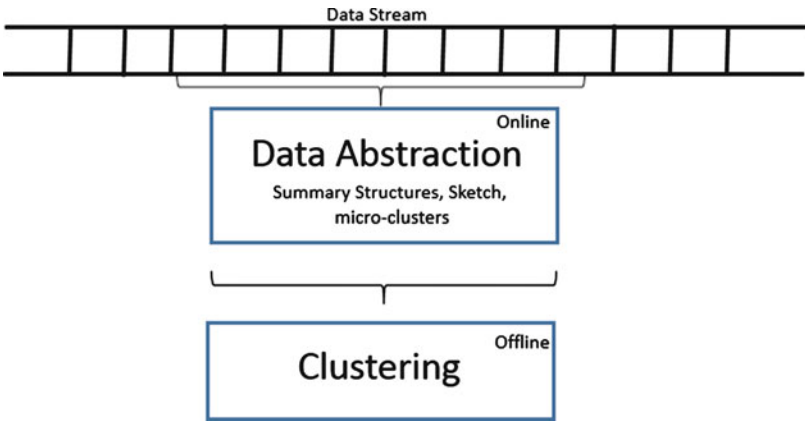
(i) provide timely results by performing fast and incremental processing of data objects;
(ii) rapidly adapt to changing dynamics of the data, which means algorithms should detect when new clusters may appear or others disappear;
(iii) scale to the number of objects that are continuously arriving;
(iv) provide a model representation that is not only compact, but that also does not grow with the number of objects processed (notice that even a linear growth should not be tolerated);
(v) rapidly detect the presence of outliers and act accordingly; and
(vi) deal with different data types, e.g., XML trees, DNA sequences, and GPS temporal and spatial information. Although these requirements are only partially fulfilled in practice, it is instructive to keep them in mind when designing algorithms for clustering data streams.

Major clustering approaches in data stream cluster analysis include:

- *Partitioning* algorithms: construct a partition of a set of objects into k clusters, which minimize some objective function (e.g., the sum of squared distances to the centroid representative). Examples include k-means (Farnstrom et al. 2000) and k-medoids (Guha et al. 2003);
- *Micro-clustering* algorithms: divide the clustering process into two phases, where the first phase is online and summarizes the data stream in local models (micro-clusters) and the second phase generates a global cluster model from the micro-clusters. Examples of these algorithms include BIRCH (Zhang et al. 1996), CluStream (Aggarwal et al. 2003), and ClusTree (Kranen et al. 2011).

Basic Concepts

Data stream clustering algorithms can be summarized into two main steps: data summarization step and clustering step, as illustrated in Fig. 1. The online abstraction step summarizes the data stream with the help of particular data structures in order to deal with space and memory constraints of stream applications. These data structures summarize the stream in order to preserve the meaning of the original objects without the need of storing them. Among the commonly employed data structures, we highlight the feature vectors (Zhang et al. 1996; Aggarwal et al. 2003), prototype arrays (Guha et al. 2003), coreset trees (Ackermann et al. 2012), and data grids (Gama et al. 2011).



Clustering from Data Streams, Fig. 1 A generic schema for clustering data streams

A powerful idea in clustering from data streams is the concept of *cluster feature – CF*. A cluster feature, or *micro-cluster*, is a compact representation of a set of points. A CF structure is a triple (N, LS, SS), used to store the sufficient statistics of a set of points:

- N is the number of data points;
- LS is a vector, of the same dimension of data points, that store the linear sum of the N points;
- SS is a vector, of the same dimension of data points, that store the square sum of the N points.

The properties of cluster features are:

- **Incrementality**

  If a point x is added to a cluster A, the sufficient statistics are updated as follows:

$$LS_A \leftarrow LS_A + x; SS_A \leftarrow SS_A + x^2; N_A \leftarrow N_A + 1$$

- **Additivity**

  If A and B are disjoint sets, merging them is equal to the sum of their parts. The additive property allows us to merge subclusters incrementally:

$$LS_C \leftarrow LS_A + LS_B; SS_C \leftarrow SS_A + SS_B; N_C \leftarrow N_A + N_B.$$

A CF entry has sufficient information to calculate the norms

$$L_1 = \sum_{i=1}^{n} |LS_{a_i} - LS_{b_i}| \text{ and}$$

$$L_2 = \sqrt{\sum_{i=1}^{n} (LS_{a_i} - LS_{b_i})^2}$$

and basic measures to characterize a cluster:

- **Centroid**, defined as the gravity center of the cluster:

$$\vec{X}0 = \frac{LS}{N}$$

- **Radius**, defined as the average distance from member points to the centroid:

$$R = \sqrt{\frac{SS}{N} - \frac{LS}{N}^2}.$$

- **Diameter**, defined as the largest distance between member points:

$$R = \sqrt{\frac{2N \times SS - 2 \times LS^2}{N \times (N-1)}}.$$

When processing and summarizing continuously arriving stream data, the most recent observations are more important because they reflect the current state of the process generating the data. A popular approach in data stream clustering consists of defining a time window that covers the most recent data. The window models that have been used in the literature are the landmark model, sliding-window model, and damped model (Gama 2010).

Partitioning Clustering

K-means is the most widely used clustering algorithm. It constructs a partition of a set of objects into k clusters, that minimize some objective function, usually a squared error function, which imply round-shape clusters. The input parameter k is fixed and must be given in advance that limits its real applicability to streaming and evolving data.

Farnstrom et al. (2000) propose a *single-pass k-Means* algorithm. The main idea is to use a buffer where points of the dataset are kept in a compressed way. The data stream is processed in blocks. All available space on the buffer is filled with points from the stream. Using these points, find k-centers such that the sum of distances from data points to their closest center is minimized. Only the k-centroids (representing the clustering results) are retained, with the corresponding k-cluster features. Only the k-centroids (representing the clustering results) are retained, with the corresponding k-cluster features. In the next iterations, the buffer is initialized with the k-centroids, found in the previous iteration and the incoming data points from the stream. The *very fast k-means* algorithm (VFKM) (Domingos and Hulten 2001) uses the Hoeffding bound to determine the number of examples needed in each step of a k-means algorithm. VFKM runs as a sequence of k-means runs, with an increasing number of examples until the Hoeffding bound is satisfied.

Guha et al. (2003) present an analytical study on k-median clustering data streams. The proposed algorithm makes a single pass over the data stream and uses small space. It requires $O(nk)$ time and $O(n\epsilon)$ space where k is the number of centers, n is the number of points, and $\epsilon < 1$. They have proved that any k-median algorithm that achieves a constant factor approximation cannot achieve a better run time than $O(nk)$.

Micro-clustering

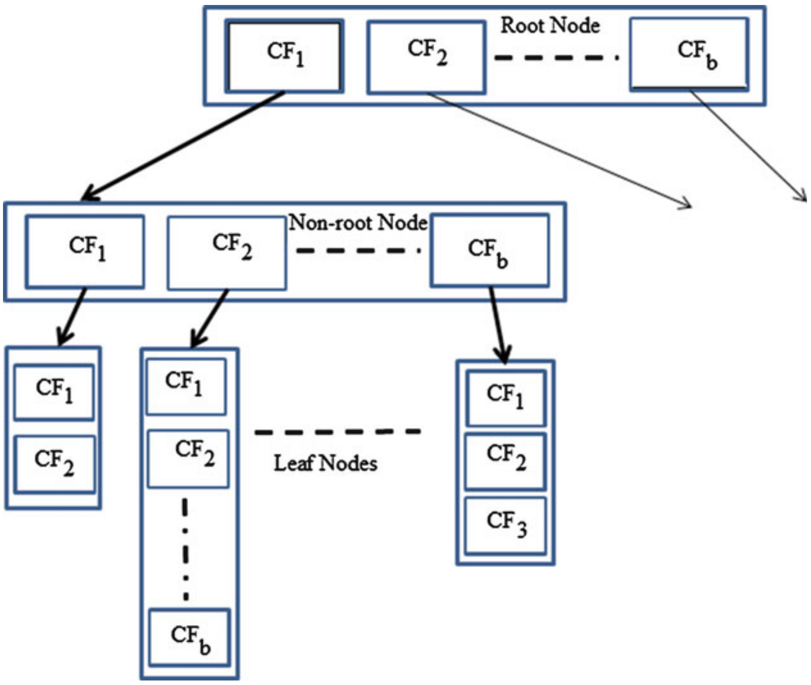
The idea of dividing the clustering process into two layers, where the first layer generates local models (micro-clusters) and the second layer generates global models from the local ones, is a powerful idea that has been used elsewhere.

The BIRCH system (Zhang et al. 1996) builds a hierarchical structure of data, the CF-tree (see Fig. 2), where each node contains a set of cluster features. These CFs contain the sufficient statistics describing a set of points in the dataset and all information of the cluster features below in the tree. The system requires two user-defined parameters: b the branch factor or the maximum number of entries in each non-leaf node and T the maximum diameter (or radius) of any CF in a leaf node. The maximum diameter T defines the examples that can be *absorbed* by a CF. Increasing T, more examples can be absorbed by a micro-cluster and smaller CF-trees are generated.

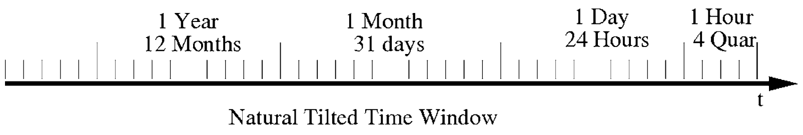
When an example is available, it traverses down the current tree from the root, till finding the appropriate leaf. At each non-leaf node, the example follows the *closest* CF path, with respect to norms L_1 or L_2. If the closest CF in the leaf cannot absorb the example, make a new CF entry. If there is no room for new leaf, split the parent node. A leaf node might be expanded due to the constrains imposed by B and T. The process consists of taking the two farthest CFs and creates two new leaf nodes. When traversing backup the CFs are updated.

Monitoring the Evolution of the Cluster Structure

The *CluStream* algorithm (Aggarwal et al. 2003) is an extension of the BIRCH system designed for data streams. Here, the CFs include temporal information: the time stamp of an example is treated as a feature. For each incoming data point, the distance to the centroids of existing CFs is computed. The data point is absorbed by an existing CF if the distance to the centroid falls



Clustering from Data Streams, Fig. 2 The clustering feature tree in BIRCH. **B** is the maximum number of CFs in a level of the tree



Clustering from Data Streams, Fig. 3 The figure presents a *natural tilted time window*. The most recent data is stored with high detail, while older data is stored in a compressed way. The degree of detail decreases with time

within the *maximum boundary* of the CF. The *maximum boundary* is defined as a factor t of the *radius* deviation of the CF; otherwise, the data point starts a new micro-cluster.

CluStream can generate approximate clusters for any user-defined time granularity. This is achieved by storing the CF at regular time intervals, referred to as snapshots. Suppose the user wants to find clusters in the stream based on a history of length h. The off-line component can analyze the snapshots stored at time t, the current time, and $(t - h)$ by using the addictive property of CF. An important problem is when to store the snapshots of the current set of micro-clusters. For example, the natural time frame stores snapshots each quarter, 4 quarters are aggregated in hours, 24 h are aggregated in days, etc. (Fig. 3). The aggregation level is domain dependent and explores the addictive property of CF. Along similar ideas, Kranen et al. (2011) present *ClusTree* that uses a weighted CF vector, which is kept into a hierarchical tree. *ClusTree* provides strategies for dealing with time constraints for anytime clustering, i.e., the possibility of interrupting the process of inserting new objects in the tree at any moment.

Tracking the Evolution of the Cluster Structure

Promising research lines are tracking change in clusters. Spiliopoulou et al. (2006) present system `MONIC`, for detecting and tracking change in clusters. MONIC assumes that a cluster is an object in a geometric space. It encompasses changes that involve more than one cluster, allowing for insights on cluster change in the whole clustering. The transition tracking mechanism is based on the degree of overlapping between the two clusters. The concept of *overlap* between two clusters, $\mathcal{X}$ and $\mathcal{Y}$, is defined as the normed number of common records weighted with the age of the records. Assume that cluster $\mathcal{X}$ was obtained at time t_1 and cluster $\mathcal{Y}$ at time t_2. The degree of overlapping between the two clusters is given by $overlap(X, Y) = \frac{\sum_{a \in X \cap Y} age(a,t_2)}{\sum_{x \in X} age(x,t_2)}$. The degree of overlapping allows inferring properties of the underlying data stream. Cluster transition at a given timepoint is a change in a cluster discovered at an earlier timepoint. MONIC considers *internal* and *external* transitions, which reflect the dynamics of the stream. Examples of cluster transitions include the cluster survives, the cluster is absorbed; a cluster disappears; and a new cluster emerges.

Recommended Reading

Ackermann MR, Martens M, Raupach C, Swierkot K, Lammersen C, Sohler C (2012) Streamkm++: a clustering algorithm for data streams. ACM J Exp Algorithmics 17:1

Aggarwal C, Han J, Wang J, Yu P (2003) A framework for clustering evolving data streams. In: Proceedings of twenty-ninth international conference on very large data bases. Morgan Kaufmann, St. Louis, pp 81–92

Domingos P, Hulten G (2001) A general method for scaling up machine learning algorithms and its application to clustering. In: Proceedings of international conference on machine learning. Morgan Kaufmann, San Francisco, pp 106–113

Farnstrom F, Lewis J, Elkan C (2000) Scalability for clustering algorithms revisited. SIGKDD Explor 2(1):51–57

Gama J (2010) Knowledge discovery from data streams. Chapman & Hall/CRC Press, Boca Raton

Gama J, Rodrigues PP, Lopes L (2011) Clustering distributed sensor data streams using local processing and reduced communication. Intell Data Anal 15(1):3–28

Guha S, Meyerson A, Mishra N, Motwani R, O'Callaghan L (2003) Clustering data streams: theory and practice. IEEE Trans Knowl Data Eng 15(3):515–528

Kranen P, Assent I, Baldauf C, Seidl T (2011) The clustree: indexing micro-clusters for anytime stream mining. Knowl Inf Syst 29(2):249–272

Silva JA, Faria E, Barros R, Hruschka E, Carvalho A, Gama J (2013) Data stream clustering: a survey. ACM Comput Surv 46(1):13

Spiliopoulou M, Ntoutsi I, Theodoridis Y, Schult R (2006) Monic: modeling and monitoring cluster transitions. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining, Philadelphia, pp 706–711

Zhang T, Ramakrishnan R, Livny M (1996) Birch: an efficient data clustering method for very large databases. In: Proceedings of ACM SIGMOD international conference on management of data. ACM Press, New York, pp 103–114

Clustering of Nonnumerical Data

▶ Categorical Data Clustering

Clustering with Advice

▶ Correlation Clustering

Clustering with Constraints

▶ Correlation Clustering

Clustering with Qualitative Information

▶ Correlation Clustering

Clustering with Side Information

▶ Correlation Clustering

Coevolution

▶ Coevolutionary Learning

Coevolutionary Computation

▶ Coevolutionary Learning

Coevolutionary Learning

R. Paul Wiegand
University of Central Florida, Orlando, FL, USA

Synonyms

Coevolution; Coevolutionary computation

Definition

Coevolutionary learning is a form of evolutionary learning (see ▸ Evolutionary Algorithms) in which the fitness evaluation is based on interactions between individuals. Since the evaluation of an individual is dependent on interactions with other evolving entities, changes in the set of entities used for evaluation can affect an individual's ranking in a population. In this sense, coevolutionary fitness is *subjective*, while fitness in traditional evolutionary learning systems typically uses an *objective* performance measure.

Motivation and Background

Ideally, coevolutionary learning systems focus on relevant areas of a search space by making adaptive changes between interacting, concurrently evolving parts. This can be particularly helpful when problem spaces are very large – infinite search spaces in particular. Additionally, coevolution is useful when applied to problems when no intrinsic objective measure exists. The interactive nature of evaluation makes them natural methods to consider for problems such as the search for game-playing strategies (Fogel 2001). Finally, some coevolutionary systems appear natural for search spaces which contain certain kinds of complex structures (Potter 1997; Stanley 2004), since search on smaller components in a larger structure can be emphasized. In fact, there is reason to believe that coevolutionary systems may be well suited for uncovering complex structures within a problem (Bucci and Pollack 2002).

Still, the dynamics of coevolutionary learning can be quite complex, and a number of pathologies often plague naïve users. Indeed, because of the subjective nature of coevolution, it can be easy to apply a particular coevolutionary learning system without a clear understanding of what kind of solution one expects a coevolutionary algorithm to produce. Recent theoretical analysis suggests that a clear concept of solution and a careful implementation of an evaluation process consistent with this concept can produce a coevolutionary system capable of addressing many problems (de Jong and Pollack 2004; Ficici 2004; Panait 2006; Wiegand 2003). Accordingly, a great deal of research in this area focuses on evaluation and progress measurement.

Structure of Learning System

Coevolutionary learning systems work in much the same way that an evolutionary learning system works: individuals encode some aspect of potential solutions to a problem, those representatives are altered during search using genetic-like operators such as mutation and crossover, and the search is directed by selecting better individuals as determined by some kind of fitness assessment. These heuristic methods gradually refine solutions by repeatedly cycling through such steps, using the ideas of heredity and survival of the fittest to produce new generations of individuals, with increased quality of solution. Just as in traditional evolutionary computation, there are many choices available to the engineer in designing such systems. The reader is referred to the chapters relating to evolutionary learning for more details.

However, there are some fundamental differences between traditional evolution and coevolution. In coevolution, measuring fitness requires evaluating the interaction between multiple individuals. Interacting individuals may reside in the same population or in different populations; the interactive nature of coevolution evokes notions of cooperation and competition in entirely new ways; the choices regarding how to best conduct evaluation of these interactions for the purposes of selection are particularly important; and there are unique coevolutionary issues surrounding representation. In addition, because of its interactive nature, the dynamics of coevolution can lead to some well-known pathological behaviors, and particularly careful attention to implementation choices to avoid such conditions is generally necessary.

Multiple Versus Single Population Approaches

Coevolution can typically be broadly classified as to whether interacting individuals reside in different populations or in the same population.

In the case of multipopulation coevolution, measuring fitness requires evaluating how individuals in one population interact with individuals in another. For example, individuals in each population may represent potential strategies for particular players of a game, they may represent roles in a larger ecosystem (e.g., predators and prey), or they may represent components that are fitted into a composite assembly with other component then applied to a problem. Though individuals in different populations interact for the purposes of evaluation, they are typically otherwise independent of one another in the coevolutionary search process.

In single population coevolution, an individual in the population is evaluated based on his or her interaction with other individuals in the same population. Such individuals may again represent potential strategies in a game, but evaluation may require them to trade off roles as to which player they represent in that game. Here, individuals interact not only for evaluation, but also implicitly compete with one another as resources used in the coevolutionary search process itself.

There is some controversy in the field as to whether this latter type qualifies as "coevolution." Evolutionary biologists often define coevolution exclusively in terms of multiple populations; however, in biological systems, fitness is always subjective, while the vast majority of computational approaches to evolutionary learning involve objective fitness assessment – and this subjective/objective fitness distinction creates a useful classification.

To be sure, there are fundamental differences between how single population and multipopulation learning systems behave (Ficici 2004). Still, single population systems that employ subjective fitness assessment behave a lot more like multipopulation coevolutionary systems than like objective fitness based evolution. Moreover, historically, the field has used the term coevolution whenever fitness assessment is based on interactions between individuals, and a large amount of that research has involved systems with only one population.

Competition and Cooperation

The terms *cooperative* and *competitive* have been used to describe aspects of coevolution learning in at least three ways.

First and less commonly, these adjectives can describe qualitatively observed behaviors of potential solutions in coevolutionary systems, the results of some evolutionary process (e.g., "tit-for-tat" strategies, Axelrod 1984).

Second, problems are sometimes considered to be inherently competitive or cooperative. Indeed, game theory provides some guidance for making such distinctions. However, since in many kinds of problems little may be known about the actual structure of the payoff functions involved, we may not actually be able to classify the problem as definitively competitive or cooperative.

The final and by far most common use of the term is to distinguish algorithms themselves. Cooperative algorithms are those in which interacting individuals succeed or fail together, while competitive algorithms are those in which individuals succeed at the expense of other individuals.

Because of the ambiguity of the terms, some researchers advocate abandoning them altogether, instead focusing distinguishing terminology on the form a potential solution takes. For example, using the term ▶ compositional coevolution to describe an algorithm designed to return a solution composed of multiple individuals (e.g., a multiagent team) and using the term ▶ test-based coevolution to describe an algorithm designed to return an individual who performs well against an adaptive set of tests (e.g., sorting network). This latter pair of terms is a slightly different, though probably more useful distinction than the cooperative and competitive terms.

Still, it is instructive to survey the algorithms based on how they have been historically classified.

Examples of competitive coevolutionary learning include simultaneously learning sorting networks and challenging data sets in a predator–prey type relationship (Hillis 1991). Here, individuals in one population representing potential sorting networks are awarded a fitness score based on how well they sort opponent data sets from the other population. Individuals in the second population represent potential data sets whose fitness is based on how well they distinguish opponent sorting networks.

Competitive coevolution has also been applied to learning game-playing strategies (Fogel 2001; Rosin and Belew 1996). Additionally, competition has played a vital part in the attempts to coevolve complex agent behaviors (Sims 1994). Finally, competitive approaches have been applied to a variety of more traditional machine learning problems, for example, learning classifiers in one population and challenging subsets of exemplars in the other (Paredis 1994).

Potter developed a relatively general framework for cooperative coevolutionary learning, applying it first to static function optimization and later to neural network learning (Potter 1997). Here, each population contains individuals representing a portion of the network, and evolution of these components occurs almost independently, in tandem with one another, interacting only to be assembled into a complete network in order to obtain fitness. The decomposition of the network can be static and a priori, or dynamic in the sense that components may be added or removed during the learning process.

Moriarty et al. take a different, somewhat more adaptive approach to cooperative coevolution of neural networks (Moriarty and Miikkulainen 1997). In this case, one population represents potential network *plans*, while a second is used to acquire node information. Plans are evaluated based on how well they solve a problem with their collaborating nodes, and the nodes receive a share of this fitness. Thus, a node is rewarded for participating more with successful plans, and thus receives fitness only indirectly.

Evaluation

Choices surrounding how interacting individuals in coevolutionary systems are evaluated for the purposes of selection are perhaps the most important choices facing an engineer employing these methods. Designing the evaluation method involves a variety of practical choices, as well as a broader eye to the ultimate purpose of the algorithm itself.

Practical concerns in evaluation include determining the number of individuals with whom to interact, how those individuals will be chosen for the interaction, and how the selection will operate on the results of multiple interactions (Wiegand 2003). For example, one might determine the fitness of an individual by pairing him or her with all other individuals in the other populations (or the same population for single population approaches) and taking the average or maximum value of such evaluations as the fitness assessment. Alternatively, one may simply use the single best individual as determined by a previous generation of the algorithm, or a combination of those approaches. Random pairings between individuals is also common. This idea can be extended to use tournament evaluation where successful individuals from pairwise interactions are promoted and further paired, assigning fitness based on how far an individual progresses in the tournament. Many of these methods have been evaluated empirically on a variety of types of problems (Angeline and Pollack 1993; Bull 1997; Wiegand 2003).

However, the designing of the evaluation method also speaks to the broader issue of how to best implement the desired ▶ solution concept, (a criterion specifying which locations in the search space are solutions and which are not) (Ficici 2004). The key to successful application of coevolutionary learning is to first elicit a clear and precise solution concept and then design an algorithm (an evaluation method in particular) that implements such a concept explicitly.

A successful coevolutionary learner capable of achieving reliable progress toward a particular solution concept often makes use of an archive of individuals and an update rule for that archive that insists the distance to a particular solution concept decrease with every change to the archive. For example, if one is interested in finding game strategies that satisfy Nash equilibrium constraints, one might consider comparing new individuals to an archive of potential individual strategies found so far that together represent a potential Nash mixed strategy (Ficici 2004). Alternatively, if one is interested in maximizing the sum of an individual's outcomes over all tests, one may likewise employ an archive of discovered tests that candidate solutions are able to solve (de Jong 2004).

It is useful to note that many coevolutionary learning problems are multiobjective in nature. That is, ▶ underlying objectives may exist in such problems, each creating a different ranking for individuals depending on the set of tests being considered during evaluation (Bucci and Pollack 2002). The set of all possible underlying objectives (were it known) is sufficient to determine the outcomes on all possible tests. A careful understanding of this can yield approaches that create ideal and minimal evaluation sets for such problems (de Jong and Pollack 2004).

By acknowledging the link between multiobjective optimization and coevolutionary learning, a variety of evaluation and selection methods based on notions of multiobjective optimization have been employed. For example, there are selection methods that use Pareto dominance between candidate solutions and their tests as their basis of comparison (Ficici 2004). Additionally, such methods can be combined with archive-based approaches to ensure monotonicity of progress toward a Pareto dominance solution concept (de Jong and Pollack 2004).

Representation

Perhaps the core representational question in coevolution is the role that an individual plays. In test-based coevolution, an individual typically represents a potential solution to the problem or a test for a potential solution, whereas in compositional coevolution individuals typically represent a candidate component for a composite or ensemble solution.

Even in test-based approaches, the true solution to the problem may be expressed as a population of individuals, rather than a single individual. The population may represent a mixed strategy while individuals represent potential pure strategies for a game. Engineers using such approaches should be clear of the form of the final solution produced by the algorithm, and that this form is consistent with the prescribed solution concept.

In compositional approaches, the key issues tend to surround about how the problem is decomposed. In some algorithms, this decomposition is performed a priori, having different populations represent explicit components of the problem (Potter 1997). In other approaches, the decomposition is intended to be somewhat more dynamic (Moriarty and Miikkulainen 1997; Potter 1997). Still more recent approaches seek to harness the potential of compositional coevolutionary systems to search open-ended representational spaces by gradually *complexifying* the representational space during the search (Stanley 2004).

In addition, a variety of coevolutionary systems have successfully dealt with some inherent pathologies by representing populations in spatial topologies, and restricting selection and interaction using geometric constraints defined by those topologies (Pagie 1999). Typically, these systems involve overlaying multiple grids of individuals, applying selection within some neighborhood in a given grid, and evaluating interactions between individuals in different grids using a similar type of cross-population neighborhood. The benefits of these systems are in part due to their ability to naturally regulate loss of diversity and spread of interaction information by explicit control over the size and shape of these neighborhoods.

Pathologies and Remedies

Perhaps the most commonly cited pathology is the so-called *loss of gradient* problem, in which one population comes to severely dominate the others, thus creating a situation in which individuals cannot be distinguished from one another. The populations become disengaged and evolutionary progress may *stall* or *drift* (Watson and Pollack 2001). Disengagement most commonly occurs when distinguishing individuals are lost in the evolutionary process (*forgetting*), and the solution to this problem typically involves somehow retaining potentially informative, though possibly inferior quality individuals (e.g., archives).

Intransitivities in the reward system can cause some coevolutionary systems to exhibit *cycling* dynamics (Watson and Pollack 2001), where reciprocal changes force the system to orbit some part of a potential search space. The remedy to this problem often involves creating coevolutionary systems that change in response to traits in several other populations. Mechanisms introduced to produce such effects include *competitive fitness sharing* (Rosin and Belew 1996).

Another challenging problem occurs when individuals in a coevolutionary systems *overspecialize* on one underlying objective at the expense of other necessary objectives (Watson and Pollack 2001). In fact, overspecialization can be seen as a form of disengagement on some subset of underlying objectives, and likewise the repair to this problem often involves retaining individuals capable of making distinctions in as many underlying objectives as possible (Bucci and Pollack 2003).

Finally, certain kinds of compositional coevolutionary learning algorithms can be prone to *relative overgeneralization*, a pathology in which components that perform reasonably well in a variety of composite solutions are favored over those that are part of an optimal solution (Wiegand 2003). In this case, it is typically possible to bias the evaluation process toward optimal values by evaluating an individual in a variety of composite assemblies and assigning the best objective value found as the fitness (Panait 2006).

In addition to pathological behaviors in coevolution, the subjective nature of these learning systems creates difficulty in measuring progress. Since fitness is subjective, it is impossible to determine whether these relative measures indicate progress or stagnation when the measurement values do not change much. Without engaging some kind of external or objective measure, it is difficult to understand what the system is really doing. Obviously, if an objective measure exists then it can be employed directly to measure progress (Watson and Pollack 2001).

A variety of measurement methodologies have been employed when objective measurement is not possible. One method is to compare current individuals against all ancestral opponents (Cliff and Miller 1995). Another predator/prey based method holds *master tournaments* between all the best predators and all the best prey found during the search (Nolfi and Floreano 1998). A similar approach suggests maintaining the best individuals from each generation in each population in a *hall of fame* for comparison purposes (Rosin and Belew 1996). Still other approaches seek to record the points during the coevolutionary search in which a new dominant individual was found (Stanley 2004). A more recent approach advises looking at the *population differential*, examining all the information from ancestral generations rather than simply selecting a biased subset (Bader and Pollack 2005). Conversely, an alternative idea is to consider how well the *dynamics* of the best individuals in different populations reflect the fundamental *best response* curves defined by the problem (Popovici 2006).

With a clear solution concept, an appropriate evaluation mechanism implementing that concept, and practical progress measures in place, coevolution can be an effective and versatile machine learning tool.

Cross-References

▸ Evolutionary Algorithms

Recommended Reading

Angeline P, Pollack J (1993) Competitive environments evolve better solutions for complex tasks. In: Forest S (ed) Proceedings of the fifth international

conference on genetic algorithms. Morgan Kaufmann, San Mateo, pp 264–270
Axelrod R (1984) The evolution of cooperation. Basic Books, New York
Bader-Natal A, Pollack J (2005) Towards metrics and visualizations sensitive to Coevolutionary failures. In: AAAI technical report FS-05-03 coevolutionary and coadaptive systems. AAAI Fall Symposium, Washington, DC
Bucci A, Pollack JB (2002) A mathematical framework for the study of coevolution. In: Poli R et al (eds) Foundations of genetic algorithms VII. Morgan Kaufmann, San Francisco, pp 221–235
Bucci A, Pollack JB (2003) Focusing versus intransitivity geometrical aspects of coevolution. In: Cantú-Paz E et al (eds) Proceedings of the 2003 genetic and evolutionary computation conference. Springer, Berlin, pp 250–261
Bull L (1997) Evolutionary computing in multi-agent environments: Partners. In: Bäck T (ed) Proceedings of the seventh international conference on genetic algorithms. Morgan Kaufmann, San Mateo, pp 370–377
Cliff D, Miller GF (1995) Tracking the red queen: measurements of adaptive progress in co-evolutionary simulations. In: Proceedings of the third European conference on artificial life. Springer, Berlin, pp 200–218
de Jong E (2004) The maxsolve algorithm for coevolution. In: Beyer H et al (eds) Proceedings of the 2005 genetic and evolutionary computation conference. ACM Press, New York, pp 483–489
de Jong E, Pollack J (2004) Ideal evaluation from coevolution. Evol Comput 12:159–192
Ficici SG (2004) Solution concepts in coevolutionary algorithms. PhD thesis, Brandeis University, Boston
Fogel D (2001) Blondie24: playing at the edge of artificial intelligence. Morgan Kaufmann, San Francisco
Hillis D (1991) Co-evolving parasites improve simulated evolution as an optimization procedure. Artificial life II, SFI studies in the sciences of complexity, vol 10. pp 313–324
Moriarty D, Miikkulainen R (1997) Forming neural networks through efficient and adaptive coevolution. Evol Comput 5:373–399
Nolfi S, Floreano D (1998) Co-evolving predator and prey robots: do "arm races" arise in artificial evolution? Artif Life 4:311–335
Pagie L (1999) Information integration in evolutionary processes. PhD thesis, Universiteit Utrecht, the Netherlands
Panait L (2006) The analysis and design of concurrent learning algorithms for cooperative multiagent systems. PhD thesis, George Mason University, Fairfax
Paredis J (1994) Steps towards co-evolutionary classification networks. In: Brooks RA, Maes P (eds) Artificial life IV, proceedings of the fourth international workshop on the synthesis and simulation of living systems. MIT Press, Cambridge, pp 359–365
Popovici E (2006) An analysis of multi-population coevolution. PhD thesis, George Mason University, Fairfax
Potter M (1997) The design and analysis of a computational model of cooperative co-evolution. PhD thesis, George Mason University, Fairfax
Rosin C, Belew R (1996) New methods for competitive coevolution. Evol Comput 5:1–29
Sims K (1994) Evolving 3D morphology and behavior by competition. In: Brooks RA, Maes P (eds) Artificial life IV, proceedings of the fourth international workshop on the synthesis and simulation of living systems. MIT Press, Cambridge, pp 28–39
Stanley K (2004) Efficient evolution of neural networks through complexification. PhD thesis, The University of Texas at Austin, Austin
Watson R, Pollack J (2001) Coevolutionary dynamics in a minimal substrate. In: Spector L et al (eds) Proceedings from the 2001 genetic and evolutionary computation conference. Morgan Kaufmann, San Francisco, pp 702–709
Wiegand RP (2003) An analysis of cooperative coevolutionary algorithms. PhD thesis, George Mason University, Fairfax

Collaborative Filtering

Collaborative Filtering (CF) refers to a class of techniques used in recommender systems, that recommend items to users that other users with similar tastes have liked in the past. CF methods are commonly sub-divided into *neighborhood-based* and *model-based* approaches. In neighborhood-based approaches, a subset of users are chosen based on their similarity to the active user, and a weighted combination of their ratings is used to produce predictions for this user. In contrast, model-based approaches assume an underlying structure to users' rating behavior, and induce predictive models based on the past ratings of all users.

Collection

► Class

Collective Classification

Galileo Namata, Prithviraj Sen, Mustafa Bilgic, and Lise Getoor
University of Maryland, College Park, MD, USA

Synonyms

Iterative classification; Link-based classification

Definition

Many real-world ▶classification problems can be best described as a set of objects interconnected via links to form a network structure. The links in the network denote relationships among the instances such that the class labels of the instances are often correlated. Thus, knowledge of the correct label for one instance improves our knowledge about the correct assignments to the other instances it connects to. The goal of collective classification is to *jointly* determine the correct label assignments of all the objects in the network.

Motivation and Background

Traditionally, a major focus of machine learning is to solve classification problems: given a corpus of documents, classify each according to its topic label; given a collection of e-mails, determine which are spam; given a sentence, determine the part-of-speech tag for each word; given a handwritten document, determine the characters, etc. However, much of the work in machine learning makes an *independent and identically distributed* (IID) assumption and focuses on predicting the class label of each instance in isolation. In many cases, however, the class labels whose values need to be determined can benefit if we know the correct assignments to related class labels. For example, it is easier to predict the topic of a webpage if we know the topics of the webpages that link to it, the chance of a particular word being a verb increases if we know that the previous word in the sentence is a noun, knowing the rest of the characters in a word can make it easier to identify an unknown character, etc. In the last decade, many researchers have proposed techniques that attempt to classify samples in a joint or collective manner instead of treating each sample in isolation and reported significant gains in classification accuracy.

Theory/Solution

Collective classification is a combinatorial optimization problem, in which we are given a set of nodes, $\mathcal{V} = \{v_1, \ldots, v_n\}$, and a neighborhood function $\mathcal{N}$, where $\mathcal{N}_i \subseteq \mathcal{V} \backslash \{v_i\}$, which describes the underlying network structure. Each node in $\mathcal{V}$ is a random variable that can take a value from an appropriate domain, $\mathcal{L} = \{l_1, \ldots, l_q\}$. $\mathcal{V}$ is further divided into two sets of nodes: $\mathcal{X}$, the nodes for which we know the correct values (observed variables), and $\mathcal{Y}$, the nodes whose values need to be determined. Our task is to label the nodes $y_i \in \mathcal{Y}$ with one of a small number of predefined labels in $\mathcal{L}$.

Even though it is only in the last decade that collective classification has entered the collective conscience of machine learning researchers, the general idea can be traced further back (Besag 1986). As a result, a number of approaches have been proposed. The various approaches to collective classification differ in the kinds of information they aim to exploit to arrive at the correct classification and their mathematical underpinnings. We discuss each in turn.

Relational Classification

Traditional classification concentrates on using the observed attributes of the instance to be classified. Relational classification (Slattery and Craven 1998) attempts to go a step further by classifying the instance using not only the instance's own attributes but also the instance's neighbors' attributes. For example, in a hypertext classification domain where we want to classify webpages, not only would we use the webpage's own words but we would also look at the webpages linking to this webpage using hyperlinks and their words to arrive at the correct class label. Results obtained using relational classification have been mixed. For example, even though there have been reports of classification accuracy gains using such techniques, in certain cases, these techniques can harm classification accuracy (Chakrabarti et al. 1998).

Iterative Collective Classification with Neighborhood Labels

A second approach to collective classification is to use the class labels assigned to the neighbor instead of using the neighbor's observed attributes. For example, going back to our hypertext classification example, instead of using the linking webpage's words, we would, in this case, use its assigned labels to classify the current webpage. Chakrabarti et al. (1998) illustrated the use of this approach and reported impressive classification accuracy gains. Neville and Jensen (2000) further developed the approach, and referred to the approach as iterative classification, and studied the conditions under which it improved classification performance (Jensen et al. 2004). Techniques for feature construction from the neighboring labels were developed and studied (Lu and Getoor 2003), along with methods that make use of *only* the label information (Macskassy and Provost 2007), as well as a variety of strategies for when to commit the class labels (McDowell et al. 2007).

Algorithm 1 depicts pseudo-code for a simple version of the iterative classification algorithm (ICA). The basic premise behind ICA is extremely simple. Consider a node $Y_i \in \mathcal{Y}$ whose value we need to determine and suppose we know the values of all the other nodes in its neighborhood $\mathcal{N}_i$ (note that $\mathcal{N}_i$ can contain both observed and unobserved variables). Then, ICA assumes that we are given a local classifier f that takes the values of $\mathcal{N}_i$ as arguments and returns a label value for Y_i from the class label set $\mathcal{L}$. For local classifiers f that do not return a class label but a goodness/likelihood value given a set of attribute values and a label, we simply choose the label that corresponds to the maximum goodness/likelihood value; in other words, we replace f with $\text{argmax}_{l \in \mathcal{L}} f$. This makes the local classifier f extremely flexible, and we can use anything ranging from a decision tree to a ▸ support vector machine (SVM). Unfortunately, it is rare in practice that we know all values in $\mathcal{N}_i$, which is why we need to repeat the process iteratively, in each iteration, labeling each Y_i using the current best estimates of $\mathcal{N}_i$ and the local classifier f and continuing to do so until the assignments to the labels stabilize.

Most local classifiers are defined as functions whose argument consists of a fixed-length vector of attribute values. A common approach to circumvent such a situation is to use an aggregation operator such as count, mode, or prop, which measures the proportion of neighbors with a given label. In Algorithm 1, we use $\vec{a}_i$ to denote the vector encoding the values in $\mathcal{N}_i$ obtained after aggregation. Note that in the first ICA iteration, all labels y_i are undefined, and to initialize them we simply apply the local classifier to the observed attributes in the neighborhood of Y_i; this is referred to as "bootstrapping" in Algorithm 1.

Researchers in collective classification (Macskassy and Provost 2007; McDowell et al. 2007; Neville and Jensen 2000) have extended the simple algorithm described above and developed a version of Gibbs sampling that is easy to imple-

Algorithm 1 Iterative classification algorithm

Iterative Classification Algorithm (ICA)

```
for each node $Y_i \in \mathcal{Y}$ do {bootstrapping}
    {compute label using only observed nodes in $\mathcal{N}_i$}
    compute $\vec{a}_i$ using only $\mathcal{X} \cap \mathcal{N}_i$
    $y_i \leftarrow f(\vec{a}_i)$
end for
repeat {iterative classification}
    generate ordering $\mathcal{O}$ over nodes in $\mathcal{Y}$
    for each node $Y_i \in \mathcal{O}$ do
        {compute new estimate of $y_i$}
        compute $\vec{a}_i$ using current assignments to $\mathcal{N}_i$
        $y_i \leftarrow f(\vec{a}_i)$
    end for
until all class labels have stabilized or a threshold number of iterations have elapsed
```

ment and faster than traditional Gibbs sampling approaches. The basic idea behind this algorithm is to assume, just like in the case of ICA, that we have access to a local classifier f that can sample for the best label estimate for Y_i given all the values for the nodes in $\mathcal{N}_i$. We keep doing this repeatedly for a fixed number of iterations (a period known as "burn-in"). After that, not only do we sample for labels for each $Y_i \in \mathcal{Y}$ but we also maintain count statistics as to how many times we sampled label l for node Y_i. After collecting a predefined number of such samples, we output the best label assignment for node Y_i by choosing the label that was assigned the maximum number of times to Y_i while collecting samples.

One of the benefits of both variants of ICA is fairly simple to make use of any local classifier. Some of the classifiers used included the following: naïve Bayes (Chakrabarti et al. 1998; Neville and Jensen 2000), ▸ logistic regression (Lu and Getoor 2003), ▸ decision trees, (Jensen et al. 2004) and weighted-vote relational neighbor (Macskassy and Provost 2007). There is some evidence to indicate that discriminately trained local classifiers such as logistic regression tend to produce higher accuracies than others; this is consistent with results in other areas.

Other aspects of ICA that have been the subject of investigation include the ordering strategy to determine in which order to visit the nodes to relabel in each ICA iteration. There is some evidence to suggest that ICA is fairly robust to a number of simple ordering strategies such as random ordering, visiting nodes in ascending order of diversity of its neighborhood class labels, and labeling nodes in descending order of label confidences (Getoor 2005). However, there is also some evidence that certain modifications to the basic ICA procedure tend to produce improved classification accuracies. For example, both (Neville and Jensen 2000) and (McDowell et al. 2007) propose a strategy where only a subset of the unobserved variables are utilized as inputs for feature construction. More specifically, in each iteration, they choose the top-k most confident predicted labels and use only those unobserved variables in the following iteration's predictions, thus ignoring the less confident predicted labels. In each subsequent iteration, they increase the value of k so that in the last iteration, all nodes are used for prediction. McDowell et al. report that such a "cautious" approach leads to improved accuracies.

Collective Classification with Graphical Models

In addition to the approaches described above, which essentially focus on local representations and propagation methods, another approach to collective classification is by first representing the problem with a high-level global ▸ graphical model and then using learning and inference techniques for the graphical modeling approach to arrive at the correct classifications. These proposals include the use of both directed ▸ graphical models (Getoor et al. 2001) and undirected graphical models (Lafferty et al. 2001; Taskar et al. 2002). See ▸ statistical relational learning and Getoor and Taskar (2007) for a survey of various graphical models that are suitable for collective classification. In general, these techniques can use both neighborhood labels and observed attributes of neighbors. On the other hand, due to their generality, these techniques also tend to be less efficient than the iterative collective classification techniques.

One common way of defining such a global model uses a *pairwise Markov random field* (pairwise MRF) (Taskar et al. 2002). Let $G = (\mathcal{V}, E)$ denote a graph of random variables as before where $\mathcal{V}$ consists of two types of random variables, the unobserved variables, $\mathcal{Y}$, which need to be assigned domain values from label set $\mathcal{L}$, and observed variables $\mathcal{X}$ whose values we know (see ▸ Graphical Models). Let Ψ denote a set of *clique potentials*. Ψ contains three distinct types of functions:

- For each $Y_i \in \mathcal{Y}$, $\psi_i \in \Psi$ is a mapping $\psi_i : \mathcal{L} \rightarrow \Re_{\geq 0}$, where $\Re_{\geq 0}$ is the set of nonnegative real numbers.
- For each $(Y_i, X_j) \in E$, $\psi_{ij} \in \Psi$ is a mapping $\psi_{ij} : \mathcal{L} \rightarrow \Re_{\geq 0}$.

- For each $(Y_i, Y_j) \in E$, $\psi_{ij} \in \Psi$ is a mapping $\psi_{ij} : \mathcal{L} \times \mathcal{L} \to \Re_{\geq 0}$.

Let $\mathbf{x}$ denote the values assigned to all the observed variables in $\mathcal{V}$, and let x_i denote the value assigned to X_i. Similarly, let $\mathbf{y}$ denote any assignment to all the unobserved variables in $\mathcal{V}$, and let y_i denote a value assigned to Y_i. For brevity of notation, we will denote by ϕ_i the clique potential obtained by computing $\phi_i(y_i) = \psi_i(y_i) \prod_{(Y_i, X_j) \in E} \psi_{ij}(y_i)$. We are now in a position to define a pairwise MRF.

Definition 1 A *pairwise Markov random field* (MRF) is given by a pair $\langle G, \Psi \rangle$ where G is a graph and Ψ is a set of clique potentials with ϕ_i and ψ_{ij} as defined above. Given an assignment $\mathbf{y}$ to all the unobserved variables $\mathcal{Y}$, the pairwise MRF is associated with the probability distribution $P(\mathbf{y}|\mathbf{x}) = \frac{1}{\mathcal{Z}(\mathbf{x})} \prod_{Y_i \in \mathcal{Y}} \phi_i(y_i) \prod_{(Y_i, Y_j) \in E} \psi_{ij}(y_i, y_j)$ where $\mathbf{x}$ denotes the observed values of $\mathcal{X}$ and $\mathcal{Z}(\mathbf{x}) = \sum_{\mathbf{y}'} \prod_{Y_i \in \mathcal{Y}} \phi_i\left(y_i'\right) \prod_{(Y_i, Y_j) \in E} \psi_{ij}\left(y_i', y_j'\right)$.

Given a pairwise MRF, it is conceptually simple to extract the best assignments to each unobserved variable in the network. For example, we may adopt the criterion that the best label value for Y_i is simply the one corresponding to the highest marginal probability obtained by summing over all other variables from the probability distribution associated with the pairwise MRF. Computationally, however, this is difficult to achieve since computing one marginal probability requires summing over an exponentially large number of terms, which is why we need approximate inference algorithms. Hence, approximate inference algorithms are typically employed, the two most common being *loopy belief propagation* (LBP) and *mean-field relaxation labeling*.

Applications

Due to its general applicability, collective classification has been applied to a number of real-world problems. Foremost in this list is document classification. Chakrabarti et al. (1998) was one of the first to apply collective classification to corpora of patents linked via hyperlinks and reported that considering attributes of neighboring documents actually hurts classification performance. Slattery and Craven (1998) also considered the problem of document classification by constructing features from neighboring documents using an ▶ inductive logic programming rule learner. Yang et al. (2002) conducted an in-depth investigation over multiple datasets commonly used for document classification experiments and identified different patterns. Other applications of collective classification include object labeling in images (Hummel and Zucker 1983), analysis of spatial statistics (Besag 1986), iterative decoding (Berrou et al. 1993), part-of-speech tagging (Lafferty et al. 2001), classification of hypertext documents using hyperlinks (Taskar et al. 2002), link prediction (Getoor et al. 2002; Taskar et al. 2003b), optical character recognition (Taskar et al. 2003a), entity resolution in sensor networks (Chen et al. 2003), predicting disulfide bonds in protein molecules (Taskar et al. 2005), segmentation of 3D scan data (Anguelov et al. 2005), and classification of e-mail speech acts (Carvalho and Cohen 2005). Recently, there have also been attempts to extend collective classification techniques to the semi-supervised learning scenario (Lu and Getoor 2003b; Macskassy 2007; Xu et al. 2006).

Cross-References

▶ Decision Tree
▶ Inductive Logic Programming
▶ Learning From Structured Data
▶ Relational Learning
▶ Semi-supervised Learning
▶ Statistical Relational Learning

Recommended Reading

Anguelov D, Taskar B, Chatalbashev V, Koller D, Gupta D, Heitz G et al (2005) Discriminative learning of Markov random fields for segmentation of 3D

scan data. In: IEEE computer society conference on computer vision and pattern recognition, San Diego. IEEE Computer Society, Washington, DC
Berrou C, Glavieux A, Thitimajshima P (1993) Near Shannon limit error-correcting coding and decoding: Turbo codes. In: Proceedings of IEEE international communications conference, Geneva. IEEE
Besag J (1986) On the statistical analysis of dirty pictures. J R Stat Soc B-48:259–302
Carvalho V, Cohen WW (2005) On the collective classification of email speech acts. In: Special interest group on information retrieval, Salvador. ACM
Chakrabarti S, Dom B, Indyk P (1998) Enhanced hypertext categorization using hyperlinks. In: International conference on management of data, Seattle. ACM, New York
Chen L, Wainwright M, Cetin M, Willsky A (2003) Multitarget multisensor data association using the tree-reweighted max-product algorithm. In: SPIE Aerosense conference, Orlando
Getoor L (2005) Link-based classification. In: Advanced methods for knowledge discovery from complex data. Springer, New York
Getoor L, Taskar B (eds) (2007) Introduction to statistical relational learning. MIT, Cambridge
Getoor L, Segal E, Taskar B, Koller D (2001) Probabilistic models of text and link structure for hypertext classification. In: Proceedings of the IJCAI workshop on text learning: beyond supervision, Seattle
Getoor L, Friedman N, Koller D, Taskar B (2002) Learning probabilistic models of link structure. J Mach Learn Res 3:679–707
Hummel R, Zucker S (1983) On the foundations of relaxation labeling processes. IEEE Trans Pattern Anal Mach Intell 5:267–287
Jensen D, Neville J, Gallagher B (2004) Why collective inference improves relational classification. In: Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining, Seattle. ACM
Lafferty JD, McCallum A, Pereira FCN (2001) conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the international conference on machine learning, Washington, DC. Morgan Kaufmann, San Francisco
Lu Q, Getoor L (2003a) Link based classification. In: Proceedings of the international conference on machine learning, Washington, DC. AAAI
Lu Q, Getoor L (2003b) Link-based classification using labeled and unlabeled data. In: ICML workshop on the continuum from labeled to unlabeled data in machine learning and data mining, Washington, DC
Macskassy S, Provost F (2007) Classification in networked data: a toolkit and a univariate case study. J Mach Learn Res 8:935–983
Macskassy SA (2007) Improving learning in networked data by combining explicit and mined links. In: Proceedings of the twenty-second AAAI conference on artificial intelligence, Vancouver. AAAI
McDowell LK, Gupta KM, Aha DW (2007) Cautious inference in collective classification. In: Proceedings of the twenty-second AAAI conference on artificial intelligence, Vancouver. AAAI
Neville J, Jensen D (2007) Relational dependency networks. J Mach Learn Res 8:653–692
Neville J, Jensen D (2000) Iterative classification in relation data. In: Workshop on statistical relational learning. AAAI
Slattery S, Craven M (1998) Combining statistical and relational methods for learning in hypertext domains. In: International conferences on inductive logic programming, Madison. Springer, London
Taskar B, Abbeel P, Koller D (2002) Discriminative probabilistic models for relational data. In: Proceedings of the annual conference on uncertainty in artificial intelligence, Edmonton. Morgan Kauffman, San Francisco
Taskar B, Guestrin C, Koller D (2003a) Max-margin Markov networks. In: Neural information processing systems. MIT, Cambridge
Taskar B, Wong MF, Abbeel P, Koller D (2003b) Link prediction in relational data. In: Natural information processing systems. MIT, Cambridge
Taskar B, Chatalbashev V, Koller D, Guestrin C (2005) Learning structured prediction models: a large margin approach. In: Proceedings of the international conference on machine learning, Bonn. ACM, New York
Xu L, Wilkinson D, Southey F, Schuurmans D (2006) Discriminative unsupervised learning of structured predictors. In: Proceedings of the international conference on machine learning, Pittsburgh. ACM, New York
Yang Y, Slattery S, Ghani R (2002) A study of approaches to hypertext categorization. J Intell Inf Syst 18(2–3):219–241

Commercial Email Filtering

▶ Text Mining for Spam Filtering

Committee Machines

▶ Ensemble Learning

Community Detection

▶ Group Detection

Comparable Corpus

A comparable corpus (pl. corpora) is a document collection composed of two or more disjoint subsets, each written in a different language, such that documents in each subset are on a same topic as the documents in the others. The prototypical example of a comparable corpora is a collection of newspaper article written in different languages and reporting about the same events: while they will not be, strictly speaking, the translation of one another, they will share most of the semantic content. Some methods for cross-language text mining rely, totally or partially, on the statistical properties of comparable corpora.

Comparison Training

▶ Preference Learning

Competitive Coevolution

▶ Test-Based Coevolution

Competitive Learning

A *Competitive learning* is an ▶ artificial neural network learning process where different neurons or processing elements compete on who is allowed to learn to represent the current input. In its purest form competitive learning is in the so-called winner-take-all networks where only the neuron that best represents the input is allowed to learn. Since all neurons learn to better represent the kinds of inputs they already are good at representing, they become specialized to represent different kinds of inputs. For vector-valued inputs and representations, the input becomes quantized to the unit having the closest representation (model), and the representations are adapted to minimize the representation error using stochastic gradient descent.

Competitive learning networks have been studied as models of how receptive fields and feature detectors, such as orientation-selective visual neurons, develop in neural networks. The same process is at work in online ▶ K-means clustering, and variants of it in ▶ Self-Organizing Maps (SOM) and the EM algorithm of mixture models.

Complex Adaptive System

▶ Complexity in Adaptive Systems

Complexity in Adaptive Systems

Jun He
Aberystwyth University, Aberystwyth, UK

Abstract

The complexity in adaptive systems is classified into two types: internal complexity for model complexity and external complexity for data complexity. As an application, the two concepts are put into the background of learning and are used to explain statistical learning.

Synonyms

Adaptive system; Complex adaptive system

Definition

An adaptive system, or complex adaptive system, is a special case of complex systems, which is able to adapt its behavior according to changes in its environment or in parts of the system itself. In this way, the system can improve its performance through a continuing interaction with its environment. The concept of complexity in an adaptive system is used to analyze the interactive relationship between the system and its environment, which can be classified into two types: internal complexity for model complexity and external complexity for data complexity. The internal complexity is defined by the amount of input, information, or energy that the system receives from its environment. The external complexity refers to the complexity of how the system represents these inputs through its internal process.

Motivation and Background

Adaptive systems range from natural systems to artificial systems (Holland 1992, 1995; Mitchell 1992). Examples of natural systems include ant colonies, ecosystem, the brain, neural network and immune system, cell, and developing embryo; examples of artificial systems include the stock market, social system, manufacturing businesses, and human social group-based endeavor in a cultural and social system such as political parties or communities. All these systems have a common feature: they can adapt to their environment.

An adaptive system is adaptive in that way it has the capacity to change its internal structure for adapting the environment. It is complex in the sense that it interacts with their environment. The interaction between an adaptive system and its environment is dynamic and nonlinear. Complexity emerges from the interaction among the system and environment and the elements of the system, where the emergent macroscopic patterns are more complex than the sum of the these low-level (microscopic) elements encompassed in the system. Understanding the evolution and development of adaptive systems still faces many mathematical challenges (Levin 2003).

The concepts of external and internal complexity are used to analyze the relation between an adaptive system and its environment. The description given below is based on Jürgen Jost's work (Jost 2004), which introduced these two concepts and applied the theoretical framework to the construction of learning models, e.g., to design neural network architectures. In the following, the concepts are mainly applied to analyze the interaction between the system and its environment. The interaction among individual elements of the system is less discussed; however, the concepts can be explored in that situation too.

Theory

Adaptive System, Environment, and Regularities

The environment of an adaptive system is more complex than the system itself and its changes cannot be completely predictable for the system. However, the changes of the environment are not purely random and noisy; there exist regularities in the environment. An adaptive system can recognize these regularities; and depending on these regularities, the system will express them through its internal process in order to adapt to the environment.

The input that an adaptive system receives or extracts from its environment usually includes two parts: one is the part with regularities and another is that appears random to the system. The part of regularities is useful and meaningful. An adaptive system will represent these regularities by internal processes. But the part of random input is useless, and even at the worst it will be detrimental for an adaptive system. However, it will depend on the adaptive system's internal model of the external environment for how to determine which part of input is meaningful and regular and which part is random and devoid of meaning and structure.

An adaptive system will translate the external regularities into its internal ones, and only the regularities are useful to the system. The system tries to extract regularities as many as possible and to represent these regularities as efficiently as possible in order to make optimal use of its capacity.

The notions of external complexity and internal complexity are used to investigate these two complementary aspects conceptually and quantitatively. In terms of these notions, an adaptive system aims to increase their external complexity and reduce their internal complexity.

The two processes operate on their own time scale but are intricately linked and mutually dependent on each other. For example, the internal complexity will be only reduced if the external complexity is fixed. Under fixed inputs received from the external environment, an adaptive system can represent these inputs systems more efficiently and optimize its internal structure. If the external complexity is increased, e.g., additional new input is required to handle by the system, then it is necessary to increase its internal complexity.

The increase of internal complexity may occur through the creation of redundancy in the existing adaptive system, e.g., to duplicate some internal structures and then enable the system to handle more external input. Once the input is fixed, the adaptive then will represent the input as efficiently as possible and reduce the internal input. The decrease of internal complexity can be achieved through discarding some input as meaningless and irrelevant, e.g., leaving some regularities out, for the purpose.

Since the inputs relevant for the systems are those which can be reflected in the internal model, the external complexity is not equivalent to the amount of raw data received from the environment. In fact, it is only relevant to the inputs which can be processed in the internal model or observations in some adaptive systems. Thus the external complexity ultimately is decided by the internal model constructed by the system.

External and Internal Complexity

External complexity means data complexity, which is used to measure the amount of input received from the environment for the system to handle and process. Such a complexity can be measured by entropy in the term of information theory.

Internal complexity is model complexity, which is used to measure the complexity of a model for representing the input or information received by the system.

The aim of the adaptive system is to obtain an efficient model as simple as possible, with the capacity to handle as much input as possible. On the one hand, the adaptive system will try to maximize its external complexity and then to adapt to its environment in a maximal way and, on the other hand, to minimize its internal complexity and then to construct a model to process the input in the most efficient way.

These two aims sometimes seem conflicting, but such a conflict can be avoided when these two processes operate on different time scales. If given a model, the system will organize the input data and try to increase its ability to deal input from its environment and then increase its external complexity. If given the input, conversely, it tries to simplify its model which represents that input and thus to decrease the internal complexity. The meaning of the input is relevant to the time scale under investigation. On a short time scale, for example, the input may consist of individual signals, but on a long time scale, it will be a sequence of signals, which satisfies a probability distribution. A good internal model tries to express regularities in the input sequence, rather than several individual signals. And the decrease of internal complexity will happen on this time scale.

A formal definition of the internal and external complexity concepts is based on the concept of entropy from statistical mechanics and information theory. Given a model θ, the system can model data as with $X(\theta) = (X_1, \cdots, X_k)$, which is assumed to have an internal probability distribution $P(X(\theta))$ so that entropy can be computed. The external complexity is defined by

$$-\sum_{i=1}^{k} P(X_i(\theta)) \log_2 P(X_i(\theta)). \tag{1}$$

An adaptive system tries to maximize the above external complexity.

The probability distribution $P(X(\theta))$ is for quantifying the information value of the data $X(\theta)$. The value of information can be described in other approaches, e.g., the length of the representation of the data in the internal code of the system (Rissanen 1989/1998). In this case, the optimal coding is a consequence of the minimization of internal complexity, and then the length of the representation of data $X_i(\theta)$ behaves like $\log_2 P(X(\theta))$ (Rissanen 1989/1998).

On a short time scale, for a given model θ, the system tries to increase the amount of meaningful input information $X(\theta)$. On a long time scale, when the input is given, e.g., when the system has gathered a set of input on a time scale with a stationary probability distribution of input patterns Ξ, then the model should be improved to handle the input as efficiently as possible and reduce the complexity of model. This complexity, or internal complexity, is defined by

$$-\sum_{i=1}^{k} P(\Xi_i \mid \theta) \log_2 P(\Xi_i \mid \theta) - \log_2 P(\theta), \quad (2)$$

with respect to the model θ.

If Rissanen's minimum description length principle (Rissanen 1989/1998) is applied to the above formula, then the optimal model will satisfy the following variation problem:

$$\min_{\theta} (-\log_2 P(\Xi \mid \theta) - \log_2 P(\theta)) . \quad (3)$$

Here in the above minimization problem, there are two objectives to minimize. The first term is to measure how efficiently the model represents or encodes the data and the second one to how complicated the model is. In computer science, this latter term corresponds to the length of the program required to encode the model.

The concepts of external and internal complexity can be applied into a system divided into subsystems. In this case, some internal part of the original whole system will become external to a subsystem. Thus the internal input of a subsystem consists of original external input and also input from the rest of the system, i.e., other subsystems.

Application: Learning

The discussion of these two concepts, external and internal complexity, can be put into the background of learning. In statistical learning theory (Vapnik 1998), the criterion for evaluating a learning process is the expected prediction error of future data by the model based on training data set with partial and incomplete information. The task is to construct a probability distribution drawn from an a priori specific class for representing the distribution underlying the input data received. Usually if a higher error is produced by a model on the training data, then a higher error will be expected on the future data. The error will depend on two factors: one is the accuracy of the model on the training data set and another is the simplicity of the model itself. The description of the data set can be split into two parts, the regular part, which is useful in construct the model, and the random part, which is a noise to the model.

The learning process fits very well into the theory framework of internal and external complexity. If the model is too complicated, it will bring the risk of over-fitting the training dada. In this case, some spurious or putative regularity is incorporated into the model, which will not appear in the future data. The model should be constrained within some model class with bounded complexity. This complexity in this context of statistical learning theory is measured by the Vapnik-Chervonenkis dimension (Vapnik 1998). Under the simplest form of statistical learning theory, the system aims at finding a representation with smallest error in a class with given complexity constraints; and then the model should minimize the expected error on future data and also over-fitting error.

The two concepts of over-fitting and leaving out regularities can be distinguished in the following sense. The former is caused by the noise in the data, i.e., the random part of the data, and this leads to putative regularities, which will not appear in the future data. The latter, leaving out regularities, means the system can forgo some part of regularities in the data, or it is possible to make data compression. Thus leaving out regularities can be used to simplify the model and reduce the internal complexity. However, a problem is still waiting for answer here, that is, what regularities in the data set are useful for data compression and also meaningful for future prediction and what parts are random to the model.

The internal complexity is the model complexity. If the internal complexity is chosen too small, then the model does not have enough capacity to represent all the important features of the data set. If the internal complexity is too large, on the other hand, then the model doesn't represent the data efficiently. The internal complexity is preferably minimized under appropriate constraints on the adequacy of the representation of data. This is consistent with Rissanen's principle of minimum description length (Rissanen 1989/1998), to represent the given data set in the most efficient way. Thus a good model is both to simplify the model itself and to represent the data efficiently.

The external complexity is the data complexity which should be large to represent the input accurately. This is related to Jaynes' principle of maximizing the ignorance (Jaynes 1957), where a model for representing data should have the maximal possible entropy under the constraint that all regularities can be reproduced. In this way, putative regularities could be eliminated in the model. However, this principle should be applied with some conditions; as argued by Gell-Mann and Lloyd (1996), it cannot eliminate the essential regularities in the data, and an overly complex model should be avoided.

For some learning system, only a selection of data is gathered and observed by the system. Thus a middle term, observation, is added between model and data. The concept of observation refers to the extraction of value of some specific quantity from a given data or data pool. What a system can observe depends on its internal structure and its general model of the environment. The system doesn't have direct access to the raw data, but through constructing a model of the environment solely on the basis of the values of its observation.

For such kind of learning system, Jaynes' principle (Jaynes 1957) is still applicable for increasing the external complexity. For the given observation made on a data set, the maximum entropy representation should be selected. However, this principle is still subject to the modification of Gell-Mann and Lloyd (1996) to the principle, where the model should not lose the essential regularities observed in the data.

In contrast, the observations should be selected to reduce the internal complexity. Given a model, if the observation can be made on a given data set, then these observations should be selected so as to minimize the resulting entropy of the model, with the purpose of minimizing the uncertainty left about the data. Thus it leads to reduce complexity.

In most cases, the environment is dynamic, i.e., the data set itself can be varied, then the external complexity should be maximized again. Thus the observation should be chosen for maximal information gain extracted from the data to increase the external complexity. Jaynes' principle (Jaynes 1957) can be applied as the same as in previous discussion. But in a longer time scale, when the input reaches some stationary distribution, the model should be simplified to reduce its internal complexity.

Recommended Reading

Gell-Mann, M and Lloyd, S (1996) Information measures, effective complexity, and total information. Complexity 2(1):44–52

Holland J (1992) Adaptation in natural and artificial systems. MIT Press, Cambridge

Holland J (1995) Hidden order: how adaptation builds complexity. Addison-Wesley, Redwood City

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

Jost J (2004) External and internal complexity of complex adaptive systems. Theory Biosci 123(1):69–88

Levin S (2003) Complex adaptive systems: exploring the known, the unknown and the unknowable. Bull Am Math Soc 40(1):3–19

Rissanen J (1989/1998) Stochastic complexity in statistical inquiry. World Scientific, Singapore

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

Mitchell W. M (1992) Complexity: the emerging science at the edge of order and chaos. Simon and Schuster, New York

Complexity of Inductive Inference

Sanjay Jain[1] and Frank Stephan[2]
[1]School of Computing, National University of Singapore, Singapore, Singapore
[2]Department of Mathematics, National University of Singapore, Singapore, Singapore

Definition

In ▸ inductive inference, the complexity of learning can be measured in various ways: by the number of hypotheses issued in the worst case until the correct hypothesis is found, by the number of data items to be consumed or to be memorized in order to learn in the worst case, by the Turing degree of oracles needed to learn the class under a certain criterion, and by the intrinsic complexity which is – like the Turing degrees in recursion theory – a way to measure the complexity of classes by using reducibilities between them.

Detail

We refer the reader to the article ▸ Inductive Inference for basic definitions in inductive inference and the notations used below. Let $\mathbb{N}$ denote the set of natural numbers. Let $\varphi_0, \varphi_1, \ldots$ denote a fixed acceptable numbering of the partial-recursive functions (Rogers 1967). Let $W_i = \mathrm{domain}(\varphi_i)$.

Mind Changes and Anomalies

The first measure of complexity of learning can be considered as the number of mind changes needed before the learner converges to its final hypothesis in the **TxtEx** model of learning. The number of mind changes by a learner M on a text T can be counted as card $(\{m : ? \neq \mathrm{M}(T[m]) \neq \mathrm{M}(T[m+1])\})$. A learner M **TxtEx**$_n$ learns a class $\mathcal{L}$ of languages if M **TxtEx** learns $\mathcal{L}$ and for all $L \in \mathcal{L}$, for all texts T for L, M makes at most n mind changes on T. **TxtEx**$_n$ is defined as the collection of language classes which can be **TxtEx**$_n$ identified (see Case and Smith (1983) for details).

Consider the class of languages $\mathcal{L}_n = \{L : \mathrm{card}(L) \leq n\}$. It can be shown that $\mathcal{L}_{n+1} \in \mathbf{TxtEx}_{n+1} - \mathbf{TxtEx}_n$.

Now consider anomalous learning. A class $\mathcal{C}$ is $\mathbf{TxtEx}_b^a$ learnable if there is a learner, which makes at most b mind changes (where $b = *$ denotes that the number of mind changes is finite on each text for a language in the class, but not necessarily bounded by a constant) and whose final hypothesis is allowed to make up to a errors (where $a = *$ denotes finitely many errors). For these learning criteria, we get a two-dimensional hierarchy on what can be learnt. Let $\mathcal{C}_n = \{f : \varphi_{f(0)} =^n f\}$. For a total function f, let $L_f = \{\langle x, f(x)\rangle : x \in \mathbb{N}\}$, where $\langle \cdot, \cdot \rangle$ denotes a computable pairing function: a bijective mapping from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. Let $\mathcal{L}_{\mathcal{C}} = \{L_f : f \in \mathcal{C}\}$. Then, one can show that $\mathcal{L}_{\mathcal{C}_{n+1}} \in \mathbf{TxtEx}_0^{n+1} - \mathbf{TxtEx}^n$. Similarly, if we consider the class $\mathcal{S}_n = \{f : \mathrm{card}(\{m : f(m) \neq f(m+1)\}) \leq n\}$, then one can show that $\mathcal{L}_{\mathcal{S}_{n+1}} \in \mathbf{TxtEx}_{n+1}^0 - \mathbf{TxtEx}_n^*$ (we refer the reader to Case and Smith (1983) for a proof of the above).

Data and Time Complexity

Wiehagen (1986) considered the complexity of the number of data needed for learning. Regarding time complexity, one should note the result by Pitt (1989) that any **TxtEx**-learnable class of languages can be **TxtEx**-learnt by a learner that has time complexity (with respect to the size of the input) bounded by a linear function. This result is achieved by a delaying trick, where the learner just repeats its old hypothesis unless it has enough time to compute its later hypothesis. This seriously effects what one can say about time complexity of learning. One proposal made by Daley and Smith (1986) is to consider the total time used by the learner until its sequence of hypotheses converges, resulting in a possibly more reasonable measure of time in the complexity of learning.

Iterative and Memory-Bounded Learning

Another measure of complexity of learning can be considered when one restricts how much past data a learner can remember. Wiehagen (1976) introduced the concept of *iterative* learning in which the learner cannot remember any past data. Its new hypothesis is based only on its previous conjecture and the new datum it receives. In other words, there exists a recursive function F such that $\mathrm{M}(T[n+1]) = F(\mathrm{M}(T[n]), T(n))$, for all texts T and for all n. Here, $\mathrm{M}(T[0])$ is some fixed value, say the symbol "?" which is used by the learner to denote the absence of a reasonable conjecture. It can be shown that being iterative restricts the learning capacity of learners. For example, let $L_e = \{2x : x \in \mathbb{N}\}$ and let $\mathcal{L} = \{L_e\} \cup \{\{S \cup \{2n + 1\}\} : n \in \mathbb{N}, S \subseteq L_e$, and $\max(S) \leq n\}$; then $\mathcal{L}$ can be shown to be **TxtEx** learnable but not iteratively learnable.

Memory-bounded learning (see Lange and Zeugmann 1996) is an extension of memory-limited learning, where the learner is allowed to memorize up to some fixed number of elements seen in the past. Thus, M is an m-memory-bounded learner if there exists a function mem and two computable functions mF and F such that, for all texts T and all n:

- $mem(T[0]) = \emptyset$;
- $M(T[n+1]) = F(M(T[n]), mem(T[n]), T(n+1))$;
- $mem(T[n+1]) = mF(M(T[n]), mem(T[n]), T(n+1))$;
- $mem(T[n+1]) - mem(T[n]) \subseteq \{T(n+1)\}$;
- $\mathrm{card}(mem(T[n])) \leq m$.

It can be shown that the criteria of inference based on **TxtEx** learning by m-memory-bounded learners form a proper hierarchy.

Besides memorizing some past elements seen, another way to address this issue is by giving feedback to the learner (see Case et al. 1999) on whether some element has appeared in the past data. A feedback learner is an iterative learner, which is additionally allowed to query whether certain elements appeared in earlier data. An n-feedback learner is allowed to make n such queries at each stage (when it receives the new input datum). Thus, M is an m-feedback learner if there exist computable functions Q and a F such that, for all texts T and all n:

- $Q(M(T[n]), T(n))$ is defined and is a set of m elements
- If $Q(M(T[n]), T(n)) = (x_1, x_2, \ldots, x_m)$, then $M(T[n+1]) = F(M(T[n]), T(n), y_1, y_2, \ldots, y_m)$, where $y_i = 1$ iff $x_i \in \mathrm{ctnt}(T[n])$.

Again, it can be shown that allowing more feedback gives greater learning power, and thus one can get a hierarchy based on the amount of feedback allowed.

Complexity of Final Hypothesis

Another possibility on complexity of learning is to consider the complexity or size of the final grammar output by the learner. Freivalds (1975) considered the case when the final program/grammar output by the learner is minimal: that is, there is no smaller index that accepts/generates the same language. He showed that this severely restricts the learning capacity of learners. Not only that, the learning capacity depends on the acceptable programming system chosen, unlike the case for most other criteria of learning such as **TxtEx** or **TxtBc**, which are independent of the acceptable programming system chosen. In particular, there are acceptable programming systems in which only classes containing finitely many infinite languages can be learnt using minimal final grammars (see Freivalds 1975; Jain and Sharma 1993). Chen (1982) considered a modification of such a paradigm where one considers convergence to nearly minimal grammars rather than minimal. That is, instead of requiring that the final grammars are minimal, one requires that they are within a recursive function h of minimal. Here h may depend on the class being learnt. Chen showed that this allows one to have the criteria of minimal learnability to be independent of the acceptable programming system chosen. However, one can show that some simple classes are not minimally learnable. An example of such a class is the class $\mathcal{L}_\mathcal{C}$ which is derived from $\mathcal{C} = \{f : \forall^\infty x[f(x) = 0]\}$, the class of all functions which are almost everywhere 0.

Intrinsic Complexity

Another way to consider complexity of learning is to consider relative complexity in a way similar to how one considers Turing reductions in computability theory. Such a notion is called intrinsic complexity of the class. This was first considered by Freivalds et al. (1995) for function learning. Jain and Sharma (1996) considered it for language learning, and the following discussion is from there.

An *enumeration operator* (see Rogers 1967), Θ, is an algorithmic mapping from SEQ into SEQ such that the following two conditions are satisfied:

- for all $\sigma, \tau \in$ SEQ, if $\sigma \subseteq \tau$, then $\Theta(\sigma) \subseteq \Theta(\tau)$;
- for all texts T, $\lim_{n\to\infty} |\Theta(T[n])| = \infty$.

By extension, we think of Θ as also mapping texts to texts such that $\Theta(T) = \bigcup_n \Theta(T[n])$. Furthermore, we define $\Theta(L) = \{\mathrm{ctnt}(\Theta(T)) : T$ is a text for $L\}$. Intuitively, $\Theta(L)$ denotes the set of languages to whose texts Θ maps texts of L. The reader should note the overloading of this notation because the type of the argument to Θ could be a sequence, a text, or a language.

One says that a sequence of grammars $g_0, g_1, \ldots$ is an acceptable **TxtEx** sequence for L if the sequence of grammars converges to a grammar for L.

$\mathcal{L}_1 \leq_{\mathrm{weak}} \mathcal{L}_2$ if there are two operators Θ and Ψ such that for all $L \in \mathcal{L}_1$, for all texts T for L, $\Theta(T)$ is a text for some $L' \in \mathcal{L}_2$ such that if $g_0, g_1, \ldots$ is an acceptable **TxtEx** sequence for L', then $\Psi(g_0, g_1, \ldots)$ is an acceptable **TxtEx** sequence for L.

Note that different texts for the same language L may be mapped by Θ to texts for different languages in $\mathcal{L}_2$ above. If we require that different texts for L are mapped to texts for the same language L' in $\mathcal{L}_2$, then we get a stronger notion of reduction called strong reduction: $\mathcal{L}_1 \leq_{\mathrm{strong}} \mathcal{L}_2$ if $\mathcal{L}_1 \leq_{\mathrm{weak}} \mathcal{L}_2$ and for all $L \in \mathcal{L}_1$, $\Theta(L)$ contains only one language, where Θ is as in the definition for $\leq_{\mathrm{weak}}$ reduction.

It can be shown that *FIN* is a complete class for **TxtEx** identification with respect to $\leq_{\mathrm{weak}}$ reduction (see Jain and Sharma 1996). Interestingly, it was shown that the class of pattern languages (Angluin 1980), the class $SD = \{L : W_{\min(L)} = L\}$, and the class $COINIT = \{\{x : x \geq n\} : n \in \mathbb{N}\}$ are all equivalent under $\leq_{\mathrm{strong}}$. Let *code* be a bijective mapping from nonnegative rational numbers to natural numbers. Then, one can show that the class RINIT $= \{\{code(x) : 0 \leq x \leq r, x$ is a rational number$\} : 0 \leq r \leq 1, r$ is a rational number $\}$ is $\leq_{\mathrm{strong}}$ complete for **TxtEx** (see Jain et al. 2001).

Interestingly every finite directed acyclic graph can be embedded into the $\leq_{\mathrm{strong}}$ degree structure (Jain and Sharma 1997a). On the other hand, the degree structure is non-dense in the sense that there exist classes $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $\mathcal{L}_1 <_{\mathrm{strong}} \mathcal{L}_2$, but for any class $\mathcal{L}$ such that $\mathcal{L}_1 \leq_{\mathrm{strong}} \mathcal{L} \leq_{\mathrm{strong}} \mathcal{L}_2$, either $\mathcal{L}_1 \equiv_{\mathrm{strong}} \mathcal{L}$ or $\mathcal{L} \equiv_{\mathrm{strong}} \mathcal{L}_2$. A similar result holds for $\leq_{\mathrm{weak}}$ reducibility (see Jain and Sharma 1997a).

Interesting connections between learning of elementary formal systems (Shinohara 1994), intrinsic complexity, and ordinal mind changes (Freivalds and Smith 1993) were shown in Jain and Sharma (1997b).

Learning Using Oracles

Another method to measure complexity of learning is to see how powerful an oracle (given to the learning machine) has to be to make a class learnable. It can be shown that an oracle A permits to explanatorily learn the class of all recursive functions iff A is high (Adleman and Blum 1991). Furthermore, an oracle is trivial, that is, does not give additional learning power for explanatory learning of function classes iff the oracle has 1-generic Turing degree and is Turing reducible to the halting problem (Slaman and Solovay 1991). The picture is a bit different in the general case of learning languages. For every oracle A, there is an oracle B and a class, which is **TxtEx** learnable using the oracle B but not using the oracle A (Jain and Sharma 1993). Note that there are also classes of languages like Gold's class of all finite languages plus the set of natural numbers which are not **TxtEx** learnable using any oracle. Furthermore, for oracles above the halting problem, **TxtEx** learning and **TxtBc** learning using these oracles coincide.

Acknowledgements Sanjay Jain was supported in part by NUS grant numbers C252-000-087-001, R146-000-

181-112, R252-000-534-112. Frank Stephen was supported in part by NUS grant numbers R146-000-181-112, R252-000-534-112.

Recommended Reading

Adleman L, Blum M (1991) Inductive inference and unsolvability. J Symb Log 56:891–900
Angluin D (1980) Finding patterns common to a set of strings. J Comput Syst Sci 21:46–62
Case J, Smith CH (1983) Comparison of identification criteria for machine inductive inference. Theor Comput Sci 25:193–220
Case J, Jain S, Lange S, Zeugmann T (1999) Incremental concept learning for bounded data mining. Inf Comput 152(1):74–110
Chen K-J (1982) Tradeoffs in inductive inference of nearly minimal sized programs. Inf Control 52:68–86
Daley RP, Smith CH (1986) On the complexity of inductive inference. Inf Control 69:12–40
Freivalds R (1975) Minimal Gödel numbers and their identification in the limit. Lect Not Comput Sci 32:219–225
Freivalds R, Smith CH (1993) On the role of procrastination in machine learning. Inf Comput 107(2):237–271
Freivalds R, Kinber E, Smith CH (1995) On the intrinsic complexity of learning. Inf Comput 123:64–71
Jain S, Sharma A (1993) On the non-existence of maximal inference degrees for language identification. Inf Process Lett 47:81–88
Jain S, Sharma A (1994) Program size restrictions in computational learning. Theor Comput Sci 127:351–386
Jain S, Sharma A (1996) The intrinsic complexity of language identification. J Comput Syst Sci 52:393–402
Jain S, Sharma A (1997a) The structure of intrinsic complexity of learning. J Symb Log 62:1187–1201
Jain S, Sharma A (1997b) Elementary formal systems, intrinsic complexity and procrastination. Inf Comput 132:65–84
Jain S, Kinber E, Wiehagen R (2001) Language learning from texts: degrees of intrinsic complexity and their characterizations. J Comput Syst Sci 63:305–354
Lange S, Zeugmann T (1996) Incremental learning from positive data. J Comput Syst Sci 53:88–103
Pitt L (1989) Inductive inference, DFAs, and computational complexity. In: Analogical and inductive inference, second international workshop (AII 1989). LNAI, vol 397. Springer, Heidelberg, pp 18–44
Rogers H (1967) Theory of recursive functions and effective computability. McGraw-Hill, New York (Reprinted, MIT Press 1987)
Shinohara T (1994) Rich classes inferable from positive data: length–bounded elementary formal systems. Inf Comput 108:175–186
Slaman TA, Solovay R (1991) When oracles do not help. In: Proceedings of the fourth annual workshop on computational learning theory, Santa Cruz. Morgan Kaufmann, pp 379–383
Wiehagen R (1976) Limes-Erkennung rekursiver Funktionen durch spezielle Strategien. J Inf Process Cybern EIK 12:93–99
Wiehagen R (1986) On the complexity of effective program synthesis. In: Jantke K (ed) Analogical and inductive inference. Proceedings of the international workshop. LNCS, vol 265. Springer, pp 209–219

Compositional Coevolution

Synonyms

Cooperative coevolution

Definition

A coevolutionary system constructed to learn composite solutions in which individuals represent different candidate components and must be evaluated together with other individuals in order to form a complete solution. Though not precisely the same as *cooperative coevolution*, there is a significant overlap.

Cross-References

▶ Coevolutionary Learning

Computational Complexity of Learning

Sanjay Jain[1] and Frank Stephan[2]
[1]School of Computing, National University of Singapore, Singapore, Singapore
[2]Department of Mathematics, National University of Singapore, Singapore, Singapore

Definition

Measures of the complexity of learning have been developed for a number of purposes including

▶ Inductive Inference, ▶ PAC Learning, and ▶ Query-Based Learning. The complexity is usually measured by the largest possible usage of resources that can occur during the learning of a member of a class. Depending on the context, one measures the complexity of learning either by a single number/ordinal for the whole class or by a function in a parameter n describing the complexity of the target to be learned. The actual measure can be the number of mind changes, the number of queries submitted to a teacher, the number of wrong conjectures issued, the number of errors made, or the number of examples processed until learning succeeds. In addition to this, one can equip the learner with an oracle and determine the complexity of the oracle needed to perform the learning process. Alternatively, in complexity theory, instead of asking for an NP-complete oracle to learn a certain class, the result can also be turned into the form "this class is unlearnable unless RP = NP" or something similar. (Here RP is the class of decision problems solvable by a randomized polynomial time algorithm, and NP is the class of decision problems solvable by a nondeterministic polynomial time algorithm, and both algorithms never give "yes" answer for an instance of the problem with "no" answer.)

Detail

In ▶ PAC Learning, one usually asks how many examples are needed to learn the concept, where the number of examples needed mainly depends on the Vapnik-Chervonenkis dimension of the class to be learned, the error permitted, and the confidence required. Furthermore, for certain classes of finite Vapnik-Chervonenkis dimension, learnability can still fail when the learner is required to be computable in polynomial time; hence, there is, besides the dimension, also a restriction stemming from the computational complexity of problems such as the complexity of finding concepts consistent with all data observed so far.

For ▶ Query-Based Learning, one common criterion to be looked at is the number of queries made during the learning process. If a class contains 2^n different $\{0,1\}$-valued functions f and one is required to learn the class using membership queries, that is, by asking queries of the form whether $f(x) = 0$ or $f(x) = 1$, then there is a function f on which the learner needs at least n queries until it knows which of the given functions f is; for some classes consisting of 2^n functions, the number of queries needed can be much worse – as much as $2^n - 1$. A well-known result of Angluin is that one can learn the class of all regular languages with polynomially many equivalence and membership queries measured with respect to the number of states of the smallest deterministic finite automaton accepting the language to be learned. Further research has been done dealing with which query algorithms can be implemented by a polynomial time learner and which need for polynomial time learning, in addition to the teacher informing on the target concept, also some oracle supplying information that cannot be computed in polynomial time. See the entry ▶ Query-Based Learning for an overview of these results.

For ▶ Inductive Inference, most complexity measures are measures applying to the overall class and not just a parameterized version. When learning the class of all sets with up to n elements, the learner might first issue the conjecture $\emptyset$ and then revise (up to n times) its hypothesis when a new datum is observed; such a measure is called the mind change complexity of learning. Mind change complexity has been generalized to measure the complexity by recursive ordinals or the notation of these. Furthermore, one can measure the long-term memory of past data observed either by a certain number of examples remembered or by the number of bits stored on a tape describing the long-term memory of the learner. Besides these quantitative notions, a further frequently studied question is the following: Which oracles support the learning process in a way that some classes become learnable using the oracle, but are unlearnable without using any oracle? An example of such a type of result is that the class of all recursive functions can be learned if and only if the learner has access to a high oracle, that is, an oracle that permits to compute a function which dominates (i.e., grows faster than) every recursive function. See the entry ▶ Complexity of Inductive Inference for more information.

Acknowledgements Sanjay Jain was supported in part by NUS grant numbers C252-000-087-001, R146-000-181-112, R252-000-534-112. Frank Stephen was supported in part by NUS grant numbers R146-000-181-112, R252-000-534-112.

Computational Discovery of Quantitative Laws

▶ Equation Discovery

Concept Drift

Claude Sammut[1] and Michael Harries[2]
[1]The University of New South Wales, Sydney, NSW, Australia
[2]Citrix Labs, Advanced Products Group, North Ryde, NSW, Australia

Synonyms

Context-sensitive learning; Learning with hidden context

Definition

Concept drift occurs when the values of hidden variables change over time. That is, there is some unknown context for ▶ concept learning and when that context changes, the learned concept may no longer be valid and must be updated or relearned.

Motivation and Background

Prediction in real-world domains is complicated by potentially unstable phenomena that are not known in advance to the learning system. For example, financial market behavior can change dramatically with changes in contract prices, interest rates, inflation rates, budget announcements, and political and world events. Thus, concept definitions that may have been learned in one context become invalid in a new context. This *concept drift* can be due to changes in context and is often directly reflected by one or more attributes. When changes in context are not reflected by any known attributes they can be said to be *hidden*. Hidden changes in context cause problems for any predictive approach that assumes concept stability.

Structure of the Learning System

Machine learning approaches can be broadly categorized as either ▶ batch Learning or ▶ incremental learning. Batch systems learn off-line by examining a large collection of instances *en masse* and form a single concept. Incremental systems evolve and change a concept definition as new observations are processed (Schlimmer and Granger 1986a; Aha et al. 1991; Kolter and Maloof 2003).

The most common approach to learning in domains with hidden changes in context has been to use an incremental learning approach in which the importance of older items is progressively decayed. A popular implementation of this, originally presented in Kubat (1989), is to use a window of recent instances from which concept updates are derived. Other examples of this approach include Widmer and Kubat (1996), Kubat and Widmer (1995), Kilander and Jansson (1993), and Salganicoff (1993). Swift adaptation to changes in context can be achieved by dynamically varying the window size in response to changes in accuracy and concept complexity (Widmer and Kubat 1996).

There are many domains in which the context can be expected not only to change but for earlier contexts to hold again at some time in the future. That is, contexts can repeat in domains such as financial prediction, dynamic control, and underrepresented data mining tasks. In these domains, prediction accuracy can be improved by storing knowledge about past contexts for reuse. FLORA3 (Widmer and Kubat 1993) addresses domains in which contexts recur by storing and retrieving concepts that appear stable as the learner traverses the series of input data.

In many situations, there is no constraint to learn incrementally. For example, many organizations maintain large data bases of historical data that are suitable for data mining. These data bases may hold instances that belong to a number of contexts but do not have this context explicitly recorded. Many of these data bases may incorporate time as an essential attribute, for example, financial records and stock market price data. Interest in mining datasets of this nature suggests the need for systems that can learn global concepts and are sensitive to changing and hidden contexts. Systems such as FLORA3 also imply that an off-line recognition of stable concepts can be useful for ▸ on-line prediction.

An alternative to on-line learning for domains with hidden changes in context is to examine the data *en masse* in an attempt to directly identify concepts associated with stable, hidden contexts. Some potential benefits of such an approach are:

1. Context specific (known as local) concepts could be used as part of a multiple model online predictive system.
2. Local concepts could be verified by experts, or used to improve domain understanding.
3. A model of the hidden context could be induced using context characteristics such as context duration, order, and stability. The model could also use existing attributes and domain feedback if available.
4. Stable contexts identified could be used as target characteristics for selecting additional attributes from the outside world as part of an iterative data mining process.

Splice (Harries et al. 1998) is a ▸ meta-learning system that implements a context sensitive batch learning approach. Splice is designed to identify intervals with stable hidden context, and to induce and refine local concepts associated with hidden contexts.

Identifying Context Change

In many domains with hidden changes in context, time can be used to differentiate hidden contexts. Most machine learning approaches to these domains do not explicitly represent time as they assume that current context can be captured by focusing on recent examples. The implication is that hidden context will be reflected in contiguous intervals of time. For example, an attempt to build a system to predict changes in the stock market could produce the following ▸ decision tree:

$$\begin{array}{llllll}
\text{Year} & > 2002 & & & & \\
\quad\text{Year} < 2005 & & & & & \\
\qquad\qquad\text{Attribute A} & = & \text{true}: & \text{Market Rising} \\
\qquad\qquad\text{Attribute A} & = & \text{false}: & \text{Market Falling} \\
\quad\text{Year} \geq 2005 & & & & & \\
\qquad\qquad\text{Attribute B} & = & \text{true}: & \text{Market Rising} \\
\qquad\qquad\text{Attribute B} & = & \text{false}: & \text{Market Falling}
\end{array}$$

This tree contains embedded knowledge about two intervals of time: in one of which, 2002–2004, attribute A is predictive; in the other, 2005 onward, attribute B is predictive. As time (in this case, year) is a monotonically increasing attribute, future classification using this decision tree will only use attribute B. If this domain can be expected to have recurring hidden context, information about the prior interval of time could be valuable.

The decision tree in the example above contains information about changes in context. We define context as:

> Context is any attribute whose values are largely independent but tend to be stable over contiguous intervals of another attribute known as the environmental attribute.

The ability of decision trees to capture context is associated with the fact that decision tree algorithms use a form of context-sensitive feature selection (CSFS). A number of machine learning algorithms can be regarded as using CSFS including decision tree algorithms (Quinlan 1993), rule induction algorithms (Clark and Niblett 1989), and ▸ ILP systems (Quinlan 1990). All of these systems produce concepts containing local information about context.

When contiguous intervals of time reflect a hidden attribute or context, we call time the environmental attribute. The environmental attribute is not restricted to time alone as it could be any ordinal attribute over which instances of a hidden context are liable to be contiguous. There is also no restriction, in principle, to one dimension. Some alternatives to time as environmental attributes are dimensions of space, and space–time combinations.

Given an environmental attribute, we can utilize a CSFS machine learning algorithm to gain information on likely hidden changes in context. The accuracy of the change points found will be dependent upon at least hidden context duration, the number of different contexts, the complexity of each local concept, and noise.

The CSFS identified context change points can be expected to contain errors of the following types:

1. ▶ Noise or serial correlation errors. These would take the form of additional incorrect change points.
2. Errors due to the repetition of tests on time in different parts of the concept. These would take the form of a group of values clustered around the actual point where the context changed.
3. Errors of omission, change points that are missed altogether.

The initial set of identified context changes can be refined by contextual ▶ clustering.

This process combines similar intervals of the dataset, where the similarity of two intervals is based upon the degree to which a partial model is accurate on both intervals.

Recent Advances

With the increasing amount of data being generated by organizations, recent work on concept drift has focused on mining from high volume ▶ data streams (Hulten et al. 2001; Wang et al. 2003; Kolter and Maloof 2003; Mierswa et al. 2006; Chu and Zaniolo 2004; Gaber et al. 2005). Methods such as Hulten et al.'s, combine decision tree learning with incremental methods for efficient updates, thus avoiding relearning large decision trees. Koltzer and Maloof also use incremental methods combined in an ▶ ensemble.

Cross-References

▶ Decision Tree
▶ Ensemble Learning
▶ Incremental Learning
▶ Inductive Logic Programming
▶ Lazy Learning

Recommended Reading

Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. Mach Learn 6:37–66
Chu F, Zaniolo C (2004) Fast and light boosting for adaptive mining of data streams. In: Advances in knowledge discovery and data mining. Lecture notes in computer science, vol 3056, pp 282–292. Springer, Berlin/New York
Clark P, Niblett T (1989) The CN2 induction algorithm. Mach Learn 3:261–283
Clearwater S, Cheng T-P, Hirsh H (1989) Incremental batch learning. In: Proceedings of the sixth international workshop on machine learning, Ithaca. Morgan Kaufmann, pp 366–370
Domingos P (1997) Context-sensitive feature selection for lazy learners. Artif Intell Rev 11:227–253. [Aha D (ed) Special issue on lazy learning.]
Gaber MM, Zaslavsky A, Krishnaswamy S (2005) Mining data streams: a review. SIGMOD Rec 34(2):18–26
Harries M, Horn K (1996) Learning stable concepts in domains with hidden changes in context. In: Kubat M, Widmer G (eds) Learning in context-sensitive domains (workshop notes). 13th international conference on machine learning, Bari
Harries MB, Sammut C, Horn K (1998) Extracting hidden context. Mach Learn 32(2):101–126
Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: KDD'01: proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 97–106
Kilander F, Jansson CG (1993) COBBIT – a control procedure for COBWEB in the presence of concept drift. In: Brazdil PB (ed) European conference on machine learning. Springer, Berlin, pp 244–261

Kolter JZ, Maloof MA (2003) Dynamic weighted majority: a new ensemble method for tracking concept drift. In: Third IEEE international conference on data mining ICDM-2003, Melbourne. IEEE CS Press, pp 123–130
Kubat M (1989) Floating approximation in time-varying knowledge bases. Pattern Recognit Lett 10:223–227
Kubat M (1992) A machine learning based approach to load balancing in computer networks. Cybern Syst J
Kubat M (1996) Second tier for decision trees. In: Machine learning: proceedings of the 13th international conference. Morgan Kaufmann, San Francisco, pp 293–301
Kubat M, Widmer G (1995) Adapting to drift in continuous domains. In: Proceedings of the eighth European conference on machine learning. Springer, Berlin, pp 307–310
Mierswa I, Wurst M, Klinkenberg R, Scholz M, Euler T (2006) Yale: rapid prototyping for complex data mining tasks. In: KDD'06: proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 935–940
Quinlan JR (1990) Learning logical definitions from relations. Mach Learn 5:239–266
Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Mateo
Salganicoff M (1993) Density adaptive learning and forgetting. In: Machine learning: proceedings of the tenth international conference. Morgan Kaufmann, San Mateo, pp 276–283
Schlimmer JC, Granger RI Jr (1986a) Beyond incremental processing: tracking concept drift. In: Proceedings AAAI-86. Morgan Kaufmann, Los Altos, pp 502–507
Schlimmer J, Granger R Jr (1986b) Incremental learning from noisy data. Mach Learn 1(3):317–354
Turney PD (1993a) Exploiting context when learning to classify. In: Brazdil PB (ed) European conference on machine learning. Springer, Berlin, pp 402–407
Turney PD (1993b) Robust classification with context sensitive features. In: Paper presented at the industrial and engineering applicatións of artificial intelligence and expert systems, Edinburgh
Turney P, Halasz M (1993) Contextual normalization applied to aircraft gas turbine engine diagnosis. J Appl Intell 3:109–129
Wang H, Fan W, Yu PS, Han J (2003) Mining concept-drifting data streams using ensemble classifiers. In: KDD'03: proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 226–235
Widmer G (1996) Recognition and exploitation of contextual clues via incremental meta-learning. In: Saitta L (ed) Machine learning: proceedings of the 13th international workshop. Morgan Kaufmann, San Francisco, pp 525–533
Widmer G, Kubat M (1993) Effective learning in dynamic environments by explicit concept tracking. In: Brazdil PB (ed) European conference on machine learning. Springer, Berlin, pp 227–243
Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23:69–101

Concept Learning

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

Synonyms

Categorization; Classification learning

Definition

The term *concept learning* is originated in psychology, where it refers to the human ability to learn categories for object and to recognize new instances of those categories. In machine learning, concept is more formally defined as "inferring a boolean-valued function from training examples of its inputs and outputs" (Mitchell 1997).

Background

Bruner et al. (1956) published their book *A Study of Thinking*, which became a landmark in psychology and would later have a major impact on machine learning. The experiments reported by Bruner, Goodnow, and Austin were directed toward understanding a human's ability to categorize and how categories are learned.

> We begin with what seems a paradox. The world of experience of any normal man is composed of a tremendous array of discriminably different objects, events, people, impressions…But were we to utilize fully our capacity for registering the differences in things and to respond to each event encountered as unique, we would soon be overwhelmed by the complexity of our

environment…The resolution of this seeming paradox…is achieved by man's capacity to categorize. To categorize is to render discriminably different things equivalent, to group objects and events and people around us into classes…The process of categorizing involves…an act of invention…If we have learned the class "house" as a concept, new exemplars can be readily recognised. The category becomes a tool for further use. The learning and utilization of categories represents one of the most elementary and general forms of cognition by which man adjusts to his environment.

The first question that they had to deal with was that of representation: what is a concept? They assumed that objects and events could be described by a set of attributes and were concerned with how inferences could be drawn from attributes to class membership. Categories were considered to be of three types: conjunctive, disjunctive, and relational.

…when one learns to categorize a subset of events in a certain way, one is doing more than simply learning to recognise instances encountered. One is also learning a rule that may be applied to new instances. The concept or category is basically, this "rule of grouping" and it is such rules that one constructs in forming and attaining concepts.

The notion of a rule as an abstract representation of a concept influenced research in machine learning. For example, ▸ decision tree learning was used as a means of creating a cognitive model of concept learning (Hunt et al. 1966). This model later inspired Quinlan's development of ID3 (Quinlan 1983).

The learning experience may be in the form of examples from a trainer or the results of trial and error. In either case, the program must be able to represent its observations of the world, and it must also be able to represent hypotheses about the patterns it may find in those observations. Thus, we will often refer to the ▸ observation language and the ▸ hypothesis language. The observation language describes the inputs and outputs of the program and the hypothesis language describes the internal state of the learning program, which corresponds to its theory of the concepts or patterns that exist in the data.

The input to a learning program consists of descriptions of objects from the universe and, in the case of ▸ supervised learning, an output value associated with the example. The universe can be an abstract one, such as the set of all natural numbers, or the universe may be a subset of the real world. No matter which method of representation we choose, descriptions of objects in the real world must ultimately rely on measurements of some properties of those objects. These may be physical properties such as size, weight, and color or they may be defined for objects, for example, the length of time a person has been employed for the purpose of approving a loan. The accuracy and reliability of a learned concept depends on the accuracy and reliability of the measurements.

A program is limited in the concepts that it can learn by the representational capabilities of both observation and hypothesis languages. For example, if an attribute/value list is used to represent examples for an induction program, the measurement of certain attributes and not others clearly places bounds on the kinds of patterns that the learner can find. The learner is said to be *biased* by its observation language (see ▸ Language Bias). The hypothesis language also places constraints on what may and may not be learned. For example, in the language of attributes and values, relationships between objects are difficult to represent. Whereas, a more expressive language, such as first-order logic, can easily be used to describe relationships.

Unfortunately, representational power comes at a price. Learning can be viewed as a search through the space of all sentences in a language for a sentence that best describes the data. The richer the language, the larger is the search space. When the search space is small, it is possible to use "brute force" search methods. If the search space is very large, additional knowledge is required to reduce the search.

Rules, Relations, and Background Knowledge

In the early 1960s, there was no discipline called "machine learning." Instead, learning was considered to be part of "pattern recognition," which had not yet split from AI. One of the main problems addressed at that time was how to represent patterns so that they could be recognized easily. Symbolic description languages were developed to be expressive and learnable.

Banerji (1960, 1962) first devised a language, which he called a "description list," which utilized an object's attributes to perform pattern recognition. Pennypacker, a masters student of Banerji at the Case Institute of Technology, implemented the recognition procedure and also used Bruner, Goodnow, and Austin's *Conservative Focussing Strategy* to learn conjunctive concepts (Pennypacker 1963). Bruner, Goodnow, and Austin describe the strategy as follows:

> ...this strategy may be described as finding a positive instance to serve as a focus, then making a sequence of choices each of which alters but one attribute value [of the focus] and testing to see whether the change yields a positive or negative instance. Those attributes of the focus which, when changed, still yield positive instance are *not* part of the concept. Those attributes of the focus that yield negative instances when changed *are* features of the concept.

The strategy is only capable of learning *conjunctive concepts*, that is, the concept description can only consist of a simple conjunction of tests on attribute values. Recognizing the limitations of simple attribute/value representations, Banerji (1964) introduced the use of predicate logic as a description language. Thus, Banerji was one of the earliest advocates of what would, many years later, become *Inductive Logic Programming*.

In the 1970s, a series of algorithms emerged that developed concept learning further. Winston's ARCH program (Winston, Learning structural descriptions from examples. PhD Thesis, MIT Artificial Intelligence Laboratory, 1970, unpublished) was influential as one of the first widely known concept learning programs. Michalski (1973, 1983) devised the Aq family of learning algorithms that set some of the early benchmarks for learning programs. Early relational learning programs were developed by Hayes-Roth (1973), Hayes-Roth and McDermott (1977), and Vere (1975, 1977).

Banerji emphasized the importance of a description language that could "grow." That is, its descriptive power should increase as new concepts are learned. These concepts become background knowledge for future learning. A simple example from Banerji (1980) illustrates the use of background knowledge. There is a language for describing instances of a concept and another for describing concepts. Suppose we wish to represent the binary number, 10, by a left-recursive binary tree of digits "0" and "1":

$$[\text{head}:\ [\text{head}:\ 1;\ \text{tail}:\ \text{nil}];\text{tail}:0]$$

"head" and "tail" are the names of attributes. Their values follow the colon. The concepts of binary digit and binary number are defined as

$$\begin{aligned} x \in digit &\equiv x = 0 \vee x = 1 \\ x \in num &\equiv (tail(x) \in digit \\ &\quad \wedge head(x) = nil) \\ &\quad \vee (tail(x) \in digit \\ &\quad \wedge head(x) \in num) \end{aligned}$$

Thus, an object belongs to a particular class or concept if it satisfies the logical expression in the body of the description. Note that the concept above is *disjunctive*. Predicates in the expression may test the membership of an object in a previously learned concept and can express *relations* between objects. Cohen and Sammut (1982) devised a learning system based on Banerji's ideas of a growing concept description language and this was further extended by Sammut and Banerji (1986).

Concept Learning and Noise

One of the most severe drawbacks of early concept learning systems was that they assumed that data sets were not noisy. That is, all attribute values and class labels in the training data are assumed to be correct. This is unrealistic in most real applications. Thus, concept learning systems began incorporating statistical measures to minimize the effects of noise and to estimate error rates (Breiman et al. 1984; Cohen 1995; Quinlan 1986, 1993).

Learning to classify objects from training examples has gone on to become one of the central themes of machine learning research. As the robustness of classification systems has increased, they have found many applications, particularly in data mining but in a broad range of other areas.

Cross-References

- ▶ Data mining on Text
- ▶ Decision Tree
- ▶ Induction
- ▶ Inductive Logic Programming
- ▶ Learning as Search
- ▶ Relational Learning
- ▶ Rule Learning

Recommended Reading

Banerji RB (1960) An information processing program for object recognition. Gen Syst 5:117–127

Banerji RB (1962) The description list of concepts. Commun Assoc Comput Mach 5(8):426–432

Banerji RB (1964) A language for the description of concepts. Gen Syst 9:135–141

Banerji RB (1980) Artificial intelligence: a theoretical approach. North Holland, New York

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth, Belmont

Bruner JS, Goodnow JJ, Austin GA (1956) A study of thinking. Wiley, New York

Cohen WW (1995) In fast effective rule induction. In: Proceedings of the twelfth international conference on machine learning, Lake Tahoe. Morgan Kaufmann, Menlo Park

Cohen BL, Sammut CA (1982) Object recognition and concept learning with CONFUCIUS. Pattern Recognit J 15(4):309–316

Hayes-Roth F (1973) A structural approach to pattern learning and the acquisition of classificatory power. In: First international joint conference on pattern recognition, Washington, DC, pp 343–355

Hayes-Roth F, McDermott J (1977) Knowledge acquisition from structural descriptions. In: Fifth international joint conference on artificial intelligence, Cambridge, MA, pp 356–362

Hunt EB, Marin J, Stone PJ (1966) Experiments in induction. Academic, New York

Michalski RS (1973) Discovering classification rules using variable valued logic system VL1. In: Third international joint conference on artificial intelligence, Stanford, pp 162–172

Michalski RS (1983) A theory and methodology of inductive learning. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach. Tioga, Palo Alto

Mitchell TM (1997) Machine learning. McGraw-Hill, New York

Pennypacker JC (1963) An elementary information processor for object recognition. SRC No. 30-I-63-1. Case Institute of Technology

Quinlan JR (1983) Learning efficient classification procedures and their application to chess end games. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach. Tioga, Palo Alto

Quinlan JR (1986) The effect of noise on concept learning. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach, vol 2. Morgan Kaufmann, Los Altos

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Mateo

Sammut CA, Banerji RB (1986) Learning concepts by asking questions. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach, vol 2. Morgan-Kaufmann, Los Altos, pp 167–192

Vere S (1975) Induction of concepts in the predicate calculus. In: Fourth international joint conference on artificial intelligence, Tbilisi, pp 351–356

Vere SA (1977) Induction of relational productions in the presence of background information. In: Fifth international joint conference on artificial intelligence, Cambridge, MA

Conditional Random Field

A *Conditional Random Field* is a form of ▶ Graphical Model for segmenting and ▶ classifying sequential data. It is the ▶ discriminative learning counterpart to the ▶ generative learning Markov Chain model.

Recommended Reading

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

Confirmation Theory

The branch of philosophy concerned with how (and indeed whether) evidence can confirm a hypothesis, even though typically it does not entail it. A distinction is sometimes drawn between *total confirmation*: how well confirmed a hypothesis is, given all available evidence and *weight-of-evidence*: the amount of extra confirmation added to the total confirmation of a hypothesis by a particular piece of evidence. Confirmation is often measured by the probability of a hypothesis conditional on evidence.

Confusion Matrix

Kai Ming Ting
Federation University, Mount Helen, VIC, Australia

Definition

A confusion matrix summarizes the classification performance of a ▶ classifier with respect to some ▶ test data. It is a two-dimensional matrix, indexed in one dimension by the true class of an object and in the other by the class that the classifier assigns. Table 1 presents an example of confusion matrix for a three-class classification task, with the classes A, B, and C.

The first row of the matrix indicates that 13 objects belong to the class A and that 10 are correctly classified as belonging to A, two misclassified as belonging to B, and one as belonging to C.

Confusion Matrix, Table 1 An example of a three-class confusion matrix

		Assigned class		
Actual class		*A*	*B*	*C*
	A	10	2	1
	B	0	6	1
	C	0	3	8

Confusion Matrix, Table 2 The outcomes of classification into positive and negative classes

		Assigned class	
Actual class		Positive	Negative
	Positive	TP	FN
	Negative	FP	TN

A special case of the confusion matrix is often utilized with two classes, one designated the *positive* class and the other the *negative* class. In this context, the four cells of the matrix are designated as ▶ *true positives* (TP), ▶ *false positives* (FP), ▶ *true negatives* (TN), and ▶ *false negatives* (FN), as indicated in Table 2.

A number of measures of classification performance are defined in terms of these four classification outcomes.

▶ *Specificity* = ▶ *True negative rate* = TN/(TN + FP)

▶ *Sensitivity* = ▶ *True positive rate* = ▶ *Recall* = TP/ (TP + FN)

▶ *Positive predictive value* = ▶ *Precision* = TP/(TP + FP)

▶ *Negative predictive value* = TN/(TN + FN)

Conjunctive Normal Form

Bernhard Pfahringer
University of Waikato, Hamilton, New Zealand

Conjunctive normal form (CNF) is an important normal form for propositional logic. A logic formula is in conjunctive normal form if it is a single conjunction of disjunctions of (possibly negated) literals. No more nesting and no other negations are allowed. Examples are:

$$a$$
$$\neg b$$
$$a \wedge b$$
$$(a \vee \neg b) \wedge (c \vee d)$$
$$\neg a \wedge (b \vee \neg c \vee d) \wedge (a \vee \neg d)$$

Any arbitrary formula in propositional logic can be transformed into conjunctive normal form by application of the laws of distribution, De Morgan's laws, and by removing double negations. It is important to note that this process can lead to exponentially larger formulas which implies that the process in the worst case runs in exponential time. An example for this behavior is the following formula given in ▸ disjunctive normal form (DNF), which is linear in the number of propositional variables in this form. When transformed into conjunctive normal form (CNF), its size is exponentially larger.

DNF: $(a_0 \wedge a_1) \vee (a_2 \wedge a_3) \vee \ldots \vee (a_{2n} \wedge a_{2n+1})$
CNF: $(a_0 \vee a_2 \vee \ldots \vee a_{2n}) \wedge (a_1 \vee a_2 \vee \ldots \vee a_{2n}) \wedge \ldots \wedge (a_1 \vee a_3 \vee \ldots \vee a_{2n+1})$

Recommended Reading

Russell S, Norvig P (2002) Artificial intelligence: a modern approach. Prentice Hall, p 215

Connection Strength

▸ Weight

Connections Between Inductive Inference and Machine Learning

John Case[1] and Sanjay Jain[2]
[1]University of Delaware, Newark, DE, USA
[2]School of Computing, National University of Singapore, Singapore, Singapore

Abstract
Inductive inference is a branch of computational learning theory which deals with learning in the limit. Though this topic deals with mostly theoretical work, it has provided some results which can be of use to practical machine learning. Some of these works include the work multitask or context-sensitive learning, learnability of elementary formal systems, behavioral cloning, learning to coordinate, geometrical clustering, and so on. The results in these areas also often give insights into limitations of science.

Definition

Inductive inference is a theoretical framework to model learning in the limit. Here we will discuss some results in inductive inference, which have relevance to machine learning community.

The mathematical/theoretical area called ▸ inductive inference is also known as *computability theoretic learning* and *learning in the limit* (Jain et al. 1999; Odifreddi 1999) typically *but, as will be seen below, not always* involves a situation depicted in (1) just below.

$$\text{Data}\quad d_0, d_1, d_2, \ldots \xrightarrow{\text{In}} M \xrightarrow{\text{Out}} \text{Programs}\quad e_0, e_1, e_2, \ldots\ . \qquad (1)$$

Let $\mathbb{N}$ = the set of nonnegative integers. Strings, including program strings, computer reals, and other data structures, inside computers, are finite bit strings and, hence, can be coded into $\mathbb{N}$. Therefore, mathematically at least, it is without loss of mathematical generality that we sometimes use the data type $\mathbb{N}$ where standard practice would use a different type.

In (1), $d_0, d_1, d_2, \ldots$ can be, e.g., the successive values of a function $f : \mathbb{N} \to \mathbb{N}$ or the elements of a (formal) language $L \subseteq \mathbb{N}$ in some order, M is a machine, the e_i's are from some hypothesis space of programs, and, for M's *successful* learning, later e_i's exactly or approximately compute the f or L.

Such learning is *offline*: in successful cases, one comes away with programs for past and future data. For the related problem of *online extrapolation* of next values for a function f, *suitable* e_i's may *be* the values of $f(i)$'s based on having seen *strictly* prior values of f.

Detail

We will discuss the off-line case until we say otherwise. It is typical in applied machine learning to present to a learner whatever data one has and to obtain *one* corresponding program hopefully for predicting these data and future data. In inductive inference the case where only one program is output is called *one-shot* learning. More typically, in inductive inference, one allows for *mind changes*, i.e., for a succession of output programs, as one receives successively more input data, with the later programs hopefully eventually being useful for predictions. Typically, one does not get success on one's first conjecture/output program, but, rather, one may achieve success eventually or, as it is said, *in the limit* after some sequence of trial and error. It is helpful at this juncture to present a problem for which this latter approach makes more sense than the one-shot approach.

We will consider some different criteria of successful learning of f or L by M. For example, **Ex**-style criteria will require that all but finitely many of the e_i's are *syntactically* the same *and* do a reasonable job of computing the f or L. **Bc**-style criteria are more relaxed, more powerful, but less useful (Bārzdiņš 1974; Case and Lynes 1982; Case and Smith 1983): they do not require almost all e_i's be the same syntactically.

Here is a well-known regression technique from, e.g., Hildebrand (1956), for exactly "curve-fitting" polynomials. It is the method involving calculating *forward differences*. We express it as a learning machine M_0 and illustrate with its being fed an *example* data sequence generated by a cubic polynomial

$$x^3 - 2x^2 + 2x + 3. \tag{2}$$

Connections Between Inductive Inference and Machine Learning, Table 1 Example sequence and its iterated forward differences

Sequence:	3		4		7		18		43
1st Diffs:		1		3		11		25	
2nd Diffs:			2		8		14		
3rd Diffs:				6		6			

See Hildebrand (1956), for how to recover the polynomials themselves.

M_0, fed a finite data sequence of natural numbers, first looks for iterated forward differences to become (apparently) constant and then outputs a rule/program, which uses the (apparent) constant to extrapolate the data sequence for any desired prediction. For example, were M_0 given the data sequence in the top row of Table 1, it would calculate 6 to be the apparent constant after *three* differencings, so M_0 then outputs the following informal rule/program.

▶ To generate the level 0 sequence, at level 0, start with 3; at level 1, start with 1; at level 2, start with 2; add the apparent constant 6 from level 3 to get successive level 2 data items; add successive level 2 items to get successive level 1 data items; *finally*, add successive level 1 items to get as many successive level 0 data items as needed for prediction.

This program, eventually output by M_0 when its input the whole top row of Table 1, correctly predicts the elements of the cubic polynomial, on successive values in $\mathbb{N}$ – the whole sequence $3, 4, 7, 18, 43, 88, 159, \ldots$. Along the way, though, just after the first data point, M_0 thinks the apparent constant is 0; just after the second that it is 1; just after the third that it is 2; and only after more of the data points does it converge for this cubic polynomial to the apparent (and, on *this* example, actual) constant 6. In general, M_0, on a polynomial of degree m, *changes its mind* up to m times until converging to its final program (of course on $f(x) = 2^x$, M_0 never converges, and each level of forward differences is just the sequence f again.).

Hence, M_0 above **Ex**-learns, e.g., the integer polynomials $f : N \to N$, but it does not in general one-shot learn these polynomials – since the data alone do not disclose the degree of a generating polynomial.

In this entry we survey some results from inductive inference but with an eye to topics having something to say regarding or to applied machine learning. In some cases, the theoretical results lend mathematical support to preexisting empirical observations about the efficacy of known machine learning techniques. In other cases, the theoretical results provide some, typically abstract, suggestions for the machine learning practitioner. In some of these cases, some of the suggestions apparently pay off in others, intriguingly, we do not know yet.

Multitask or Context-Sensitive Learning

In empirical, applied machine learning, *multitask* or *context-sensitive learning* involves trying to learn Y by first (de Garis 1990a,b; Fahlman 1991; Thrun 1996; Thrun and Sullivan 1996; Tsung and Cottrell 1989; Waibel 1989a,b) *or* simultaneously (Caruana 1993, 1996; Matwin and Kubat 1996; Bartlmae et al. 1997; Dietterich et al. 1995; Mitchell et al. 1994; Pratt et al. 1991; Sejnowski and Rosenberg 1986) trying to learn also X – even in cases where there may be no inherent interest in learning X. There is, in many cases, an apparent *empirical* advantage in doing this for some X, Y. It can happen that Y is not apparently or easily learnable by itself but is learnable if one learns X first or simultaneously in some case X itself can be a sequence of tasks $X_1, \ldots, X_n$. Here the X_is may need to be learned sequentially or simultaneously to learn Y. For example, to teach a robot to drive a car, it is useful to train it also to predict the center of the road markings (see, e.g., Baluja and Pomerleau 1995; Caruana 1996). For another example, an experimental system to predict the value of German *Daimler* stock performed better when it was modified to track simultaneously the German stock index DAX (Bartlmae et al. 1997). The value of the Daimler stock here was the primary or target concept and the value of the DAX – a related concept – provided useful auxiliary context.

Angluin et al. (1989) shows *mathematically* that, in effect, there are (mathematical) learning scenarios for which it was *provable* that Y could not be learned without learning X *first* , and, in other scenarios (Angluin et al. 1989; Kinber et al. 1995), Y could not be learned without *simultaneously* learning X. These *mathematical* results provide a kind of evidence that the *empirical* observations as to the *apparent* usefulness of multitask or context-sensitive learning *may* not be illusionary, luck, or a mere accident of happening to use some data sets but not others.

For illustration, here is a particularly simple theoretical example needing to be learned *simultaneously* and similar to examples in Angluin et al. (1989). Let $\mathcal{R}$ be the set of all computable functions mapping $\mathbb{N}$ to $\mathbb{N}$. We use numerical names in $\mathbb{N}$ for programs. Let

$$\mathcal{S} = \{(f, g) \in \mathcal{R} \times \mathcal{R} \mid f(0) \text{ is a program for } g \wedge g(0) \text{ is a program for } f\}. \tag{3}$$

We say (p, q) is a program for $(f, g) \in \mathcal{R} \times \mathcal{R}$ iff p is a program for f and q is a program for g.

Consider a machine M which, if, as in (1), M is fed $d_0, d_1, \ldots$, but *where* each d_i is $(f(i), g(i))$, then M outputs each $e_i = (g(0), f(0))$. Clearly, M one-shot learns $\mathcal{S}$. It can be easily shown that the component f's and g's *for* $(f, g) \in \mathcal{S}$ are not *separately* even **Bc**-learnable. It is important to note that, perhaps quite unlike real-world problems, the definition of this example $\mathcal{S}$ employs a simple self-referential coding trick: useful programs are coded into values of the functions at argument zero. A number of inductive inference results have been proved by means of (sometimes more complicated) self-referential coding tricks (see, e.g., Case 1994). Bārzdiņš indirectly (see Zeugmann 1986) provided a kind of informal robustness idea in his attempt to be rid of such coding tricks in inductive inference. More formally, Fulk (1990) considered a learnability result involving a witnessing class $\mathcal{C}$ of (tuples of) functions to be *robust* iff each computable scrambling of $\mathcal{C}$ also witnesses the learnability result (the allowed computable scramblers are the *general recursive operators* of (Rogers 1967), but we omit the formal details herein.) Example: A simple shift scrambler converting each f to f', where $f'(x) = f(x+1)$, would eliminate the coding tricks just above – since the values of f at argument zero would be lost in this scrambling. Some inductive inference results hold robustly and some not (see, e.g., Fulk 1990; Jain et al. 1999, 2001; Jain 1999;

Case et al. 2000). Happily, the $\mathcal{S} \subseteq \mathcal{R} \times \mathcal{R}$ above (i.e., learnable, but its components not) can be replaced by a more complicated class $\mathcal{S}'$ that *robustly* witnesses the same result. This is better *theoretical* evidence that the empirically noticed efficacy of multitask or context-sensitive learning is not just an accident. It is residually important to note that (Jain et al. 2001) shows, though, that the computable scramblers cannot get rid of more sophisticated coding tricks they called topological. $\mathcal{S}'$ mentioned just above turns out to *employ* this latter kind of coding trick. It is hypothesized in Case et al. (2000) that nature likely employs some sophisticated coding tricks itself. For a separate informal argument about coding tricks of nature, see Case (1999). Ott and Stephan (2002) introduce a finite invariance constraint on top of robustness. This so-called hyperrobustness does destroy all coding tricks, and the result about the *theoretical* efficacy of multitask or context-sensitive learning is *not* hyperrobust. However, hyperrobustness, perhaps, leaves unrealistically sparse structure.

Final note: Machine learning is an engineering endeavor. However, philosophers of science as well as practitioners in classical scientific disciplines should likely be considering the relevance of multitask or context-sensitive inductive inference to their endeavors.

Special Cases of Inductive Logic Programming

In this section we discuss some *learning in the limit* results for *elementary formal systems (EFSs)* (Smullyan 1961). Essentially, EFSs are programs in a string rewriting system. It is well known (Arikawa et al. 1992) that EFSs are essentially (pure) logic programs over strings. Hence, the results have possible relevance for ▸ inductive logic programming (ILP) (Muggleton and De Raedt 1994; Lavrač and Džeroski 1994; Bratko and Muggleton 1995; Mitchell 1997).

First we will discuss some important special cases based on Angulin's *pattern languages* (Angluin 1980).

A *pattern language* is (by definition) one generated by all the positive length substitution instances in a *pattern*, such as,

$$abXYcbbZXa \tag{4}$$

— where the variables (for substitutions) are depicted in upper case and the constants/terminals in lower case and are from, say, the alphabet {a,b,c}. Just below is an EFS or logic program based on this example pattern.

$$abXYcbbZXa \leftarrow . \tag{5}$$

It must be understood, though, that in (5) and in the next example EFS below, only positive length strings are allowed to be substituted for the variables.

Angluin (1980) showed the **Ex**-learnability of the class of pattern languages from positive data. For these results, in the paradigm of (1) above $d_0, d_1, d_2, \ldots$ is a listing or presentation of some formal language L over a finite nonempty alphabet, and the e_i's are programs that generate languages. In particular, for Angluin's M, for L a pattern language, the e_i's are patterns, and, for each presentation of L, all but finitely many of the corresponding e_i's are the same *correct* pattern for L.

Much work has been done on the learnability of pattern languages, e.g., Salomaa (1994a,b), Case et al. (2001), and on bounded finite unions thereof, e.g., Shinohara (1983), Wright (1989), Kilpeläinen et al. (1995), Brazma et al. (1996), and Case et al. (1999).

Regarding bounded finite unions of pattern languages, an *n-pattern language* is the union of the pattern languages for some n patterns $P_1, \ldots, P_n$. Each n-pattern language is also **Ex**-learnable from positive data (see Wright 1989). An EFS or logic program corresponding to the n-patterns $P_1, \ldots, P_n$ and generating the corresponding n-pattern language is just below.

$$\begin{gathered} P_1 \leftarrow . \\ \vdots \\ P_n \leftarrow . \end{gathered}$$

Pattern language learning algorithms have been successfully applied toward some problems in molecular biology; see, e.g., Shimozono et al. (1994), Shinohara and Arikawa (1995).

Lange and Wiehagen (1991) present an interesting *iterative* (Wiehagen 1976) algorithm learning the class of pattern languages – from positive data only and with fair polynomial time constraints (for examples of fair vs. unfair polynomial time learning, see Case and Kötzing 2009). *Iterative learners* are **Ex**-learners for which each output depends only on its just prior output (if any) and the input data element currently seen. Their algorithm works in polynomial time (actually quadratic time) in the length of the latest data item and the previous hypothesis. Furthermore, the algorithm has a linear set of good examples, in the sense that if the input data contains these good examples, then the algorithm already converges to the correct hypothesis. The number of good examples needed is at most $|P| + 1$, where P is a pattern generating the data $d_0, d_1, d_2, \ldots$ for the language being learned. This algorithm may be useful in practice due to its fast run time and being able to converge quickly, *if* enough good data is available early. Furthermore, due to iterativeness, it does not need to store previous data!

Zeugmann (1998) considers *total* learning time up to convergence of the algorithm just discussed in the just prior paragraph. Note that, for arbitrary presentations, $d_0, d_1, d_2, \ldots$, of a pattern language, this time can be unbounded. In the best case, it is polynomial in the length of a generating pattern P, where $d_0, d_1, d_2, \ldots$ is based on using P to get good examples early – in fact the time taken in the best case is $\Theta(|P|^2 log_s(s + k))$, where P is the pattern, s is the alphabet size, and k is the number of variables in P. Much more interesting is the case of *average time* taken up to convergence. The probability distribution (called *uniform* by Zeugmann) considered is as follows. A variable X is replaced by a string w with probability $\frac{1}{(2s)^{|w|}}$ (i.e., all strings of length r together have probability 2^{-r}, and the distribution is uniform among strings of length r). Different variables are replaced independently of each other. In this case the average total time up to convergence is $O(2^k k^2 s |P|^2 log_s(ks))$. The main thing is that for average case on probabilistic data (as can be expected in real life, though not necessarily with this kind of uniform distribution), the algorithm converges pretty fast and computations are done efficiently.

A number of papers consider **Ex**-learning of EFSs (Krishna Rao 1996; Krishna Rao and Sattar 1998; Krishna Rao 2000, 2004, 2005) including with various bounds on the number of mind changes until syntactic convergence to correct programs (Jain and Sharma 1997, 2002). The EFSs considered are patterns, n-patterns, those with a constant bound on the length of clauses, and some with constant bounds on search trees. The mind change bounds are typically more dynamic than those given by constants: they involve counting down from finite representations (called *notations*) for infinite constructive ordinals. *An example* of this kind of bound: one can algorithmically, based on some input parameters, decide how many mind changes will be allowed. In *other* examples, the decision as to how many mind changes will be allowed can be algorithmically revised some constant number of times. It is possible that not yet created special cases of some of these algorithms could be made feasible enough for practice.

Learning Drifting Concepts

A drifting concept to be learned is one which is a moving target (see ▶ Concept Drift). In some machine learning applications, concept drift must be dealt with (Bartlett et al. 1996; Blum and Chalasani 1992; Devaney and Ram 1994; Freund and Mansour 1997; Helmbold and Long 1994; Kubat 1992; Wrobel 1994; Widmer and Kubat 1996). An inductive inference contribution is (Case et al. 2001) in which it is shown, for online *extrapolation* by computable martingale betting strategies, upper bounds on the "speed" of the moving target that permit success at all. Here success is to make unbounded amounts of "money" betting on correctness of one's extrapolations. Here is an illustrative result from

(Case et al. 2001). For the pattern languages considered in the previous section, only *positive* length strings of terminals can be substituted for a variable in an associated pattern. The (difficult to learn) *pattern languages with erasing* are just the languages obtained by also allowing the substitution of the empty string for variables in a pattern. For our example, we restrict the terminal alphabet to be {0,1}. With each pattern language with erasing L (over this terminal alphabet), we associate its characteristic function χ_L, which is 1 on terminal strings in L and 0 on those not in L. For ε denoting the empty string, and for the terminal strings in length-lexicographical order, $\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$, we would input a χ_L itself to a potential extrapolating machine as the bit string, $\chi_L(\varepsilon), \chi_L(0), \chi_L(1), \chi_L(00), \chi_L(01), \ldots$. Let $\mathcal{E}$ be the class of these characteristic functions. Pick a positive integer constant p. To model *drift with permanence* p, we imagine that a potential extrapolator for $\mathcal{E}$ receives successive bits from a member of $\mathcal{E}$ but keeps switching to the next bits of another, etc., *but* it must see at least p bits in a row of each member of $\mathcal{E}$ it sees before it can see the next bits of another. p is, then, a speed limit on drift. The result is that some suitably clever computable martingale betting strategy is successful at extrapolating $\mathcal{E}$ with drift permanence (speed limit on drift) of $p = 7$.

Behavioral Cloning

Kummer and Ott (1996) and Case et al. (2002) studied learning in the limit of winning control strategies for *closed computable games*. These games nicely model *reactive* process-control problems. Included are such example process-control games as regulating temperature of a room to be in a desired interval, forever after no more than some *fixed* number of moves between the thermostat and processes disturbing the temperature (Roughly, *closed computable games* are those so that one can tell algorithmically when one has lost. A temperature control game that requires stability forever after some *undetermined* finite number of moves is *not* a closed computable game. For a more formal treatment, see Cenzer and Remmel (1992), Maler et al. (1995), Thomas (1995), and Kummer and Ott (1996).

In machine learning, there are cases where one wants to teach a machine some motor skill possessed by human experts and where these human experts do not have access to verbalizable knowledge about how they perform expertly. Piloting an aircraft or expert operation of a swinging shipyard crane provide examples, and machine learning employs, in these cases, ▸ *behavioral cloning*, which uses direct performance data from the experts (Bain and Sammut 1999; Bratko et al. 1998; Šuc 2003).

Case et al. (2002) study the effects on learning in the limit *closed computable games* where the learning procedures also had access to the *behavioral performance* (but not the algorithms) of masters/experts at the games. For example, it is showed that, in some cases, there is better performance cloning $n + 1$ disparate masters over cloning only n. For a while it was not known *in machine learning* how to clone multiple experts even after Case et al. (2002) was known to some; however, independently of Case et al. (2002), and later, Dorian Šuc (2003) found a way to clone behaviorally more than one human expert simultaneously (for the free-swinging shipyard crane problem) – by having more than one level of feedback control, *and* he got enhanced performance from cloning the multiple experts!

Learning to Coordinate

Montagna and Osherson (1999) begin the study of learning in the limit to *coordinate* (digital) moves between at least two agents.

The machines of Montagna and Osherson (1999) are, in effect, general extrapolating devices (Montagna and Osherson 1999; Case et al. 2005). Technically, and without loss of generality of the results, we restrict the moves of each coordinator to bits, i.e., zeros and ones. *Coordination* is achieved between two coordinators iff each, reacting to the bit sequence of the other, eventually (in the limit) matches it bit for bit. Montagna and Osherson (1999) give an example of two people who show up in a park each day at one of noon (bit 0) or 6pm (bit 1); each *silently* watches the other's past behavior, and each *tries*, based on the past behavior of the other, to show up eventually exactly when the other shows up. If they manage it, they have learned to coordinate.

A *blind* coordinator is one that reacts only to the *presence* of a bit from another process, *not* to *which* bit the other process has played (Montagna and Osherson 1999).

Case et al. (2005) developed and studied the notion of probabilistically correct algorithmic coordinators. Next is a sample of theorems to the effect that just a few random bits can enhance learning to coordinate.

Theorem 1 (Case et al. 2005) *Suppose* $0 \leq p < 1$. *There exists a class of deterministic algorithmic coordinators* $\mathcal{C}$ *such that:*

(1) No *deterministic algorithmic coordinator can coordinate with all of* $\mathcal{C}$
(2) For k *chosen so that* $1-2^{-k} \geq p$, *there exists a blind,* probabilistic *algorithmic coordinator* **PM**, *such that:*
   (i) For each member of $\mathcal{C}$, **PM** *can coordinate with with probability* $1-2^{-k} \geq p$
   (ii) **PM** *is* k-memory limited *in the sense of (Osherson et al. 1986, P. 66); more specifically,* **PM** *needs to remember whether it is outputting one of its first* k *bits* — which are its only random bits *(e.g., for* $p = \frac{255}{256}$, *a mere* $k = 8$ *random bits suffice.).*

Regarding possible eventual applicability: Maye et al. (2007) cite finding deterministic chaos but *not* randomness in the behavior of *animals*. Hence, animals may not be exploiting random bits in learning anything, including to coordinate. However, one might build artifactual devices to exploit randomness, say, from radioactive decay, including, then, for enhancing learning to coordinate.

Learning Geometric Clustering

Case et al. (2006) showed that learnability in the limit of ▶ *clustering*, with or without additional information, depends strongly on geometric constraints on the shape of the clusters. In this approach the hypothesis space of possible clusters is pre-given in each setting. It was hoped to obtain thereby insight into the difficulty of clustering when the clusters are restricted to preassigned *geometrically* defined classes.

This is interestingly complementary to the *conceptual clustering* approach (see, e.g., Pitt and Reinke 1988; Mishra et al. 2004) where one restricts the possible clusters to have good "verbal" descriptions in some language.

Clustering of many of the geometric classes investigated was shown to *require* information *in addition* to a presentation, $d_0, d_1, d_2, \ldots$, of the set of points to be clustered. For example, for clusters as convex hulls of finitely many points in a rational vector space, clustering can be done – but with the number of clusters as additional information. Let $\mathcal{S}$ consist of all polygons including their interiors – in the rational two-dimensional plane *without intersections and degenerated angles*. (Attention was restricted to spaces of rationals since 1. computer reals are rationals, 2. this avoids the uncountability of the set of reals, and 3. this avoids dealing with *un*computable real points.) The class $\mathcal{S}$ *can* be clustered – but with the number of vertices of the polygons of the clusters involved as additional information.

Correspondingly, then, it was shown that the class $\mathcal{S}'$ containing $\mathcal{S}$ *together with* all such polygons but with one hole (the nondegenerate differences of two members in $\mathcal{S}$) can*not* be clustered with the number of vertices as additional information, yet $\mathcal{S}'$ can be clustered with *area* as additional information – and this even in higher dimensions and with any number of holes (Case et al. 2006).

It remains to be seen if some forms of geometrically constrained clustering can be usefully complementary to, say, conceptually/verbally constrained clustering.

Insights for Limitations of Science

We briefly treat below in some problems regarding parsimonious, refutable, and consistent hypotheses.

It is common wisdom in science that one should fit parsimonious explanations, hypotheses, or programs to data. In machine learning, this has been successfully applied, e.g., (Wallace and Dowe 1999; Wallace 2005).

Curiously, though, there are many results in inductive inference in which we see sometimes severe *degradations* of learning power caused by demanding *parsimonious* predictive programs (see, e.g., Freivalds 1975; Kinber 1977; Chen 1982; Case et al. 1996; Ambainis et al. 2004).

It is an interesting problem to resolve the seeming, likely not actual contradiction between the just prior two paragraphs.

Popper's Refutability (1962) asserts that hypotheses in science should be subject to refutation. Besides the well-known difficulties of Duhem–Quine (Harding 1976) of knowing which component hypothesis to throw out when a compound hypothesis badly fails to make correct predictions, inductive inference theorems have provided very different difficulties. Case and Smith (1983) outline cases of usefully *in*complete (hence wrong) hypothesis that cannot be refuted, and Case and Suraj (2007) (see also Case 2007) provide cases of inductively inferable higher order hypothesis not totally subject to refutation in cases where ordinary hypotheses subject to full refutation cannot be inductively inferred.

While Duhem–Quine may impact machine learning eventually, it remains to be seen about the inductive inference results of the just prior paragraph.

Requiring ▶ inductive inference procedures always to output an hypothesis in various senses *consistent* with (e.g., not ignoring) the data on which that hypothesis is based seems like mere common sense. However, from Bārzdiņš (1974a), Blum and Blum (1975), Wiehagen (1976), and Case et al. (2004), we see that strict adherence to various consistency principles can severely attenuate the learning power of inductive inference machines. Furthermore, interestingly, *even when inductive inference is polytime constrained*, we see similar counterintuitive results to the effect that a kind of consistency can strictly attenuate learning power (Wiehagen and Zeugmann 1994).

A machine learning analog might be Breiman's bagging (Breiman 1996) and random forests (Breiman 2001), where data is purposely ignored. However, in these cases, the purpose of ignoring data is to avoid overfitting to noise.

It remains to be seen, whether, in applied machine learning involving cases of practically noiseless data, one can also obtain some advantage in ignoring some consistency principles. Again the potential lesson from inductive inference is abstract and provides only a hint of something to work out in real machine learning problems.

Cross-References

▶ Behavioral Cloning
▶ Clustering
▶ Concept Drift
▶ Inductive Logic Programming

Recommended Reading

Ambainis A, Case J, Jain S, Suraj M (2004) Parsimony hierarchies for inductive inference. J Symb Logic 69:287–328

Angluin D, Gasarch W, Smith C (1989) Training sequences. Theor Comput Sci 66(3):255–272

Angluin D (1980) Finding patterns common to a set of strings. J Comput Syst Sci 21:46–62

Arikawa S, Shinohara T, Yamamoto A (1992) Learning elementary formal systems. Theor Comput Sci 95:97–113

Bain M, Sammut C (1999) A framework for behavioural cloning. In: Furakawa K, Muggleton S, Michie D (eds) Machine intelligence, vol 15. Oxford University Press, Oxford

Baluja S, Pomerleau D (1995) Using the representation in a neural network's hidden layer for task specific focus of attention. Technical report CMU-CS-95-143, School of Computer Science, CMU, May 1995. Appears in proceedings of the 1995 IJCAI

Bartlett P, Ben-David S, Kulkarni S (1996) Learning changing concepts by exploiting the structure of change. In: Proceedings of the ninth annual conference on computational learning theory, Desenzano del Garda. ACM Press, New York

Bartlmae K, Gutjahr S, Nakhaeizadeh G (1997) Incorporating prior knowledge about financial markets through neural multitask learning. In: Refenes APN, Burgess AN, Moody JE (eds) Decision technologies for computational finance. Proceedings of the fifth international conference on computational finance. Kluwer Academic, pp 425–432

Bārzdiņš J (1974a) Inductive inference of automata, functions and programs. In: Proceedings of the international congress of mathematicians, Vancouver, pp 771–776

Bārzdiņš J (1974b) Two theorems on the limiting synthesis of functions. In: Theory of algorithms and programs, vol 210. Latvian State University, Riga, pp 82–88

Blum L, Blum M (1975) Toward a mathematical theory of inductive inference. Inf Control 28:125–155

Blum A, Chalasani P (1992) Learning switching concepts. In: Proceedings of the fifth annual conference on computational learning theory, Pittsburgh. ACM Press, New York, pp 231–242

Bratko I, Muggleton S (1995) Applications of inductive logic programming. Commun ACM 38(11):65–70

Bratko I, Urbančič T, Sammut C (1998) Behavioural cloning of control skill. In: Michalski RS, Bratko I, Kubat M (eds) Machine learning and data mining: methods and applications. Wiley, New York, pp 335–351

Brazma A, Ukkonen E, Vilo J (1996) Discovering unbounded unions of regular pattern languages from positive examples. In: Proceedings of the seventh international symposium on algorithms and computation (ISAAC'96). Lecture notes in computer science, vol 1178. Springer, Berlin, pp 95–104

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Caruana R (1993) Multitask connectionist learning. In: Proceedings of the 1993 connectionist models summer school. Lawrence Erlbaum, Hillsdale, pp 372–379

Caruana R (1996) Algorithms and applications for multitask learning. In: Proceedings 13th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 87–95

Case J (1994) Infinitary self-reference in learning theory. J Exp Theor Artif Intell 6:3–16

Case J (1999) The power of vacillation in language learning. SIAM J Comput 28(6):1941–1969

Case J (2007) Directions for computability theory beyond pure mathematical. In: Gabbay D, Goncharov S, Zakharyaschev M (eds) Mathematical problems from applied logic II. New logics for the twenty-first century. International mathematical series, vol 5. Springer, New York

Case J, Kötzing T (2009) Difficulties in forcing fairness of polynomial time inductive inference. In: Gavalda R, Lugosi G, Zeugmann T, Zilles S (eds) 20th international conference on algorithmic learning theory (ALT'09). LNAI, vol 5809. Springer, Berlin, pp 263–277

Case J, Lynes C (1982) Machine inductive inference and language identification. In: Nielsen M, Schmidt E (eds) Proceedings of the 9th international colloquium on automata, languages and programming. Lecture notes in computer science, vol 140. Springer, Berlin, pp 107–115

Case J, Smith C (1983) Comparison of identification criteria for machine inductive inference. Theor Comput Sci 25:193–220

Case J, Suraj M (2007) Weakened refutability for machine learning of higher order definitions 2007. Working paper for eventual journal submission

Case J, Jain S, Kaufmann S, Sharma A, Stephan F (2001) Predictive learning models for concept drift (special issue for ALT'98). Theor Comput Sci 268:323–349

Case J, Jain S, Lange S, Zeugmann T (1999) Incremental concept learning for bounded data mining. Inf Comput 152:74–110

Case J, Jain S, Montagna F, Simi G, Sorbi A (2005) On learning to coordinate: random bits help, insightful normal forms, and competency isomorphisms (special issue for selected learning theory papers from COLT'03, FOCS'03, and STOC'03). J Comput Syst Sci 71(3):308–332

Case J, Jain S, Martin E, Sharma A, Stephan F (2006) Identifying clusters from positive data. SIAM J Comput 36(1):28–55

Case J, Jain S, Ott M, Sharma A, Stephan F (2000) Robust learning aided by context (special issue for COLT'98). J Comput Syst Sci 60:234–257

Case J, Jain S, Sharma A (1996) Machine induction without revolutionary changes in hypothesis size. Inf Comput 128:73–86

Case J, Jain S, Stephan F, Wiehagen R (2004) Robust learning – rich and poor. J Comput Syst Sci 69(2):123–165

Case J, Ott M, Sharma A, Stephan F (2002) Learning to win process-control games watching gamemasters. Inf Comput 174(1):1–19

Cenzer D, Remmel J (1992) Recursively presented games and strategies. Math Soc Sci 24:117–139

Chen K (1982) Tradeoffs in the inductive inference of nearly minimal size programs. Inf Control 52:68–86

de Garis H (1990a) Genetic programming: building nanobrains with genetically programmed neural network modules. In: IJCNN: international joint conference on neural networks, vol 3. IEEE Service Center, Piscataway, pp 511–516

de Garis H (1990b) Genetic programming: modular neural evolution for Darwin machines. In: Caudill M (ed) IJCNN-90-WASH DC; international joint conference on neural networks, vol 1. Lawrence Erlbaum Associates, Hillsdale, pp 194–197 de Garis H (1991) Genetic programming: building artificial nervous systems with genetically programmed neural network modules. In: Soušek B, The IRIS group (eds) Neural and intelligenct systems integeration: fifth and sixth generation integerated reasoning information systems, Chap. 8. Wiley, New York, pp 207–234

Devaney M, Ram A (1994) Dynamically adjusting concepts to accommodate changing contexts. In: Kubat M, Widmer G (eds) Proceedings of the ICML-96 pre-conference workshop on learning in context-sensitive domains, Bari. Journal submission

Dietterich T, Hild H, Bakiri G (1995) A comparison of ID3 and backpropogation for English text-tospeech mapping. Mach Learn 18(1):51–80

Fahlman S (1991) The recurrent cascade-correlation architecture. In: Lippmann R, Moody J, Touretzky D (eds) Advances in neural information processing systems, vol 3. Morgan Kaufmann Publishers, San Mateo, pp 190–196

Freivalds R (1975) Minimal Gödel numbers and their identification in the limit. Lecture notes in computer science, vol 32. Springer, Berlin, pp 219–225

Freund Y, Mansour Y (1997) Learning under persistent drift. In: Ben-David S, (ed) Proceedings of the third European conference on computational learning theory (EuroCOLT'97). Lecture notes in artificial intelligence, vol 1208. Springer, Berlin, pp 94–108

Fulk M (1990) Robust separations in inductive inference. In: Proceedings of the 31st annual symposium on foundations of computer science. IEEE Computer Society, St. Louis, pp 405–410

Harding S (ed) (1976) Can theories be refuted? Essays on the Duhem-Quine thesis. Kluwer Academic Publishers, Dordrecht

Helmbold D, Long P (1994) Tracking drifting concepts by minimizing disagreements. Mach Learn 14:27–46

Hildebrand F (1956) Introduction to numerical analysis. McGraw-Hill, New York

Jain S (1999) Robust behaviorally correct learning. Inf Comput 153(2):238–248

Jain S, Sharma A (1997) Elementary formal systems, intrinsic complexity, and procrastination. Inf Comput 132:65–84

Jain S, Sharma A (2002) Mind change complexity of learning logic programs. Theor Comput Sci 284(1):143–160

Jain S, Osherson D, Royer J, Sharma A (1999) Systems that learn: an introduction to learning theory, 2nd edn. MIT Press, Cambridge, MA

Jain S, Smith C, Wiehagen R (2001) Robust learning is rich. J Comput Syst Sci 62(1):178–212

Kilpeläinen P, Mannila H, Ukkonen E (1995) MDL learning of unions of simple pattern languages from positive examples. In: Vitányi P (ed) Computational learning theory, second European conference, EuroCOLT'95. Lecture notes in artificial intelligence, vol 904. Springer, Berlin, pp 252–260

Kinber E (1977) On a theory of inductive inference. Lecture notes in computer science, vol 56. Springer, Berlin, pp 435–440

Kinber E, Smith C, Velauthapillai M, Wiehagen R (1995) On learning multiple concepts in parallel. J Comput Syst Sci 50:41–52

Krishna Rao M (1996) A class of prolog programs inferable from positive data. In: Arikawa A, Sharma A (eds) Seventh international conference on algorithmic learning theory (ALT' 96). Lecture notes in artificial intelligence, vol 1160. Springer, Berlin, pp 272–284

Krishna Rao M (2000) Some classes of prolog programs inferable from positive data (Special Issue for ALT'96). Theor Comput Sci A 241:211–234

Krishna Rao M (2004) Inductive inference of term rewriting systems from positive data. In: Ben-David S, Case J, Maruoka A (eds) Algorithmic learning theory: fifteenth international conference (ALT'2004). Lecture notes in artificial intelligence, vol 3244. Springer, Berlin, pp 69–82

Krishna Rao M (2005) A class of prolog programs with non-linear outputs inferable from positive data. In: Jain S, Simon HU, Tomita E (eds) Algorithmic learning theory: sixteenth international conference (ALT'2005). Lecture notes in artificial intelligence, vol 3734. Springer, Berlin, pp 312–326

Krishna Rao M, Sattar A (1998) Learning from entailment of logic programs with local variables. In: Richter M, Smith C, Wiehagen R, Zeugmann T (eds) Ninth international conference on algorithmic learning theory (ALT'98). Lecture notes in artificial intelligence, vol 1501. Springer, Berlin, pp 143–157

Kubat M (1992) A machine learning based approach to load balancing in computer networks. Cybern Syst 23:389–400

Kummer M, Ott M (1996) Learning branches and learning to win closed recursive games. In: Proceedings of the ninth annual conference on computational learning theory, Desenzano del Garda. ACM Press, New York

Lange S, Wiehagen R (1991) Polynomial time inference of arbitrary pattern languages. New Gener Comput 8:361–370

Lavrač N, Džeroski S (1994) Inductive logic programming: techniques and applications. Ellis Horwood, New York

Maler O, Pnueli A, Sifakis J (1995) On the synthesis of discrete controllers for timed systems. In: Proceedings of the annual symposium on the theoretical aspects of computer science. LNCS, vol 900. Springer, Berlin, pp 229–242

Matwin S, Kubat M (1996) The role of context in concept learning. In: Kubat M, Widmer G (eds) Proceedings of the ICML-96 pre-conference workshop on learning in context-sensitive domains, Bari, pp 1–5

Maye A, Hsieh C, Sugihara G, Brembs B (2007) Order in spontaneous behavior. PLoS One, May 2007. http://brembs.net/spontaneous/

Mishra N, Ron D, Swaminathan R (2004) A new conceptual clustering framework. Mach Learn 56 (1–3):115–151

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

Mitchell T, Caruana R, Freitag D, McDermott J, Zabowski D (1994) Experience with a learning, personal assistant. Commun ACM 37:80–91

Montagna F, Osherson D (1999) Learning to coordinate: a recursion theoretic perspective. Synthese 118:363–382

Muggleton S, De Raedt L (1994) Inductive logic programming: theory and methods. J Logic Program 19/20:669–679

Odifreddi P (1999) Classical recursion theory, vol II. Elsivier, Amsterdam

Osherson D, Stob M, Weinstein S (1986) Systems that learn: an introduction to learning theory for cognitive and computer scientists. MIT Press, Cambridge, MA

Ott M, Stephan F (2002) Avoiding coding tricks by hyperrobust learning. Theor Comput Sci 284(1):161–180

Pitt L, Reinke R (1988) Criteria for polynomial-time (conceptual) clustering. Mach Learn 2:371–396

Popper K (1992) Conjectures and refutations: the growth of scientific knowledge. Basic Books, New York

Pratt L, Mostow J, Kamm C (1991) Direct transfer of learned information among neural networks. In: Proceedings of the 9th national conference on artificial intelligence (AAAI-91), Anaheim. AAAI press, Menlo Park

Rogers H (1987) Theory of recursive functions and effective computability. McGraw Hill, New York. (Reprinted, MIT Press, 1987)

Salomaa A (1994a) Patterns (The formal language theory column). EATCS Bull 54:46–62

Salomaa A (1994b) Return to patterns (The formal language theory column). EATCS Bull 55: 144–157

Sejnowski T, Rosenberg C (1986) NETtalk: a parallel network that learns to read aloud. Technical report JHU-EECS-86-01, Johns Hopkins University

Shimozono S, Shinohara A, Shinohara T, Miyano S, Kuhara S, Arikawa S (1994) Knowledge acquisition from amino acid sequences by machine learning system BONSAI. Trans Inf Process Soc Jpn 35:2009–2018

Shinohara T (1983) Inferring unions of two pattern languages. Bull Inf Cybern 20:83–88

Shinohara T, Arikawa A (1995) Pattern inference. In: Jantke KP, Lange S (eds) Algorithmic learning for knowledge-based systems. Lecture notes in artificial intelligence, vol 961. Springer, Berlin, pp 259–291

Smullyan R (1961) Theory of formal systems. Annals of mathematics studies, vol 47). Princeton University Press, Princeton

Šuc D (2003) Machine reconstruction of human control strategies. Frontiers in artificial intelligence and applications, vol 99. IOS Press, Amsterdam

Thomas W (1995) On the synthesis of strategies in infinite games. In: Proceedings of the annual symposium on the theoretical aspects of computer science. LNCS, vol 900. Springer, Berlin, pp 1–13

Thrun S (1996) Is learning the n-th thing any easier than learning the first? In: Advances in neural information processing systems, vol 8. Morgan Kaufmann, San Mateo

Thrun S, Sullivan J (1996) Discovering structure in multiple learning tasks: the TC algorithm. In: Proceedings of the thirteenth international conference on machine learning (ICML-96). Morgan Kaufmann, San Francisco, pp 489–497

Tsung F, Cottrell G (1989) A sequential adder using recurrent networks. In: IJCNN-89-WASHINGTON DC: international joint conference on neural networks, 18–22 June, vol 2. IEEE Service Center, Piscataway, pp 133–139

Waibel A (1989a) Connectionist glue: modular design of neural speech systems. In: Touretzky D, Hinton G, Sejnowski T (eds) Proceedings of the 1988 connectionist models summer school. Morgan Kaufmann, San Mateo, pp 417–425

Waibel A (1989b) Consonant recognition by modular construction of large phonemic time-delay neural networks. In: Touretzky DS (ed) Advances in neural information processing systems I. Morgan Kaufmann, San Mateo, pp 215–223

Wallace C (2005) Statistical and inductive inference by minimum message length. Information science and statistics. Springer, New York. Posthumously published

Wallace C, Dowe D (1999) Minimum message length and Kolmogorov complexity (special issue on Kolmogorov complexity). Comput J 42(4):123–155. http://comjnl.oxfordjournals.org/cgi/reprint/42/4/270

Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23:69–101

Wiehagen R (1976) Limes-Erkennung rekursiver Funktionen durch spezielle Strategien. Electronische Informationverarbeitung und Kybernetik 12: 93–99

Wiehagen R, Zeugmann T (1994) Ignoring data may be the only way to learn efficiently. J Exp Theor Artif Intell 6:131–144

Wright K (1989) Identification of unions of languages drawn from an identifiable class. In: Rivest R, Haussler D, Warmuth M (eds) Proceedings of the second annual workshop on computational learning theory, Santa Cruz. Morgan Kaufmann Publishers, San Mateo, pp 328–333

Wrobel S (1994) Concept formation and knowledge revision. Kluwer Academic Publishers, Dordrecht

Zeugmann T (1986) On Bārzdiņš' conjecture. In: Jantke KP (ed) Proceedings of the international workshop on analogical and inductive inference. Lecture notes in computer science, vol 265. Springer, Berlin, pp 220–227

Zeugmann T (1998) Lange and Wiehagen's pattern language learning algorithm: an average case analysis with respect to its total learning time. Ann Math Artif Intell 23:117–145

Connectivity

▶ Topology of a Neural Network

Consensus Clustering

Synonyms

Clustering aggregation; Clustering ensembles

Definition

In Consensus Clustering we are given a set of n objects V, and a set of m clusterings $\{C_1, C_2, \ldots, C_m\}$ of the objects in V. The aim is to find a single clustering C that *disagrees* least with the input clusterings, that is, C minimizes

$$D(C) = \sum_{C_i} d(C, C_j),$$

for some metric d on clusterings of V. Meilă (2003) proposed the principled *variation of information* metric on clusterings, but it has been difficult to analyze theoretically. The Mirkin metric is the most widely used, in which $d(C, C')$ is the number of pairs of objects (u, v) that are clustered together in C and apart in C', or vice versa; it can be calculated in time $O(mn)$.

We can interpret each of the clusterings C_i in Consensus Clustering as evidence that pairs ought be put together or separated. That is, w^i_{uv} is the number of C_i in which $C_i[u] = C_i[v]$ and w^-_{uv} is the number of C_i in which $C_i[u] \neq C_i[v]$. It is clear that $w^+_{uv} + w^-_{uv} = m$ and that Consensus clustering is an instance of Correlation clustering in which the w^-_{uv} weights obey the triangle inequality.

Constrained Clustering

Kiri L. Wagstaff
Pasadena, CA, USA

Definition

Constrained clustering is a semisupervised approach to ▶ clustering data while incorporating domain knowledge in the form of constraints. The constraints are usually expressed as pairwise statements indicating that two items must, or cannot, be placed into the same cluster. Constrained clustering algorithms may enforce every constraint in the solution, or they may use the constraints as guidance rather than hard requirements.

Motivation and Background

▶ Unsupervised learning operates without any domain-specific guidance or preexisting knowledge. Supervised learning requires that all training examples be associated with labels. Yet it is often the case that existing knowledge for a problem domain fits neither of these extremes. Semisupervised learning methods fill this gap by making use of both labeled and unlabeled data. Constrained clustering, a form of semisupervised learning, was developed to extend clustering algorithms to incorporate existing domain knowledge, when available. This knowledge may arise from labeled data or from more general rules about the concept to be learned.

One of the original motivating applications was noun phrase coreference resolution, in which noun phrases in a text must be clustered together to represent distinct entities (e.g., "Mr. Obama" and "the President" and "he", separate from "Sarah Palin" and "she" and "the Alaska governor"). This problem domain contains several natural rules for when noun phrases should (such as appositive phrases) or should not (such as a mismatch on gender) be clustered together. These rules can be translated into a collection of pairwise constraints on the data to be clustered.

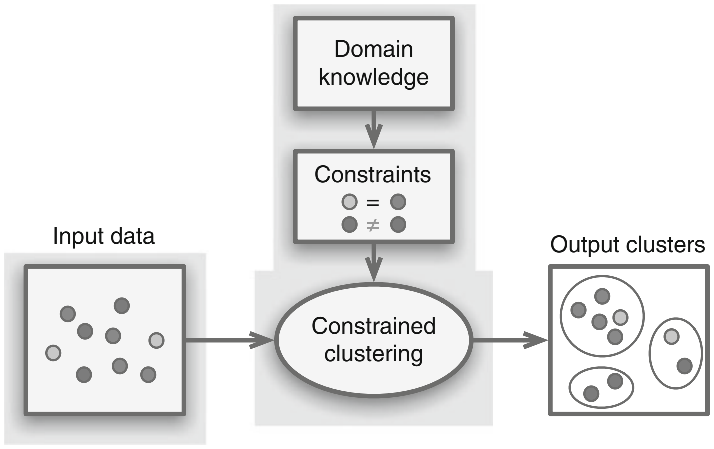
Constrained clustering algorithms have now been applied to a rich variety of domain areas, including hyperspectral image analysis, road lane divisions from GPS data, gene expression microarray analysis, video object identification, document clustering, and web search result grouping.

Structure of the Learning System

Constrained clustering arises out of existing work with unsupervised clustering algorithms. In this description, we focus on clustering algorithms that seek a partition of the data into disjoint clusters, using a distance or similarity measure to place similar items into the same cluster. Usually, the desired number of clusters, k, is specified as an input to the algorithm. The most common clustering algorithms are k-means (MacQueen 1967) and expectation maximization or EM (Dempster et al. 1977) (Fig. 1).

A constrained clustering algorithm takes the same inputs as a regular (unsupervised) clustering algorithm and also accepts a set of pairwise constraints. Each constraint is a must-link or ▶ cannot-link constraint. The must-link constraints form an equivalence relation, which permits the inference of additional transitively implied must-links as well as additional entailed cannot-link constraints between items from distinct must-link cliques. Specifying a significant number of pairwise constraints might be tedious for large data sets, so often they may be generated from a manually labeled subset of the data or from domain-specific rules.

The algorithm may interpret the constraints as hard constraints that must be satisfied in the output or as soft preferences that can be violated, if necessary. The former approach was used in the first constrained clustering algorithms, COP-COBWEB (Wagstaff and Cardie 2000) and COP-kmeans (Wagstaff et al. 2001). COP-kmeans accommodates the constraints by restricting item assignments to exclude any constraint violations. If a solution that satisfies the constraints is not found, COP-kmeans terminates without a solution. Later, algorithms such as PCK-means and MPCK-means (Bilenko et al. 2004) permitted the violation of constraints when necessary by introducing a violation penalty. This is useful when the constraints may contain noise or internal inconsistencies, which are especially relevant in real-world domains. Constrained versions of other clustering algorithms such as EM (Shental et al. 2004) and spectral clustering (Kamvar et al. 2003) also exist. Penalized probabilistic clustering (PPC) is a modified version of EM that interprets the constraints as (soft) probabilistic priors on the relationships between items (Lu and Leen 2005).



Constrained Clustering, Fig. 1 The constrained clustering algorithm takes in nine items and two pairwise constraints (one must-link and one cannot-link). The output clusters respect the specified constraints

In addition to constraining the assignment of individual items, constraints can be used to learn a better distance metric for the problem at hand (Bar-Hillel et al. 2005; Klein et al. 2002; Xing et al. 2003). Must-link constraints hint that the effective distance between those items should be low, while cannot-link constraints suggest that their pairwise distance should be high. Modifying the metric accordingly permits the subsequent application of a regular clustering algorithm, which need not explicitly work with the constraints at all. The MPCK-means algorithm fuses these approaches together, providing both constraint satisfaction and metric learning simultaneously (Basu et al. 2004; Bilenko et al. 2004).

More information about subsequent advances in constrained clustering algorithms, theory, and novel applications can be found in a compilation edited by Basu et al. (2008).

Programs and Data

The MPCK-means algorithm is available in a modified version of the Weka machine learning toolkit (Java) at http://www.cs.utexas.edu/users/ml/risc/code/.

Recommended Reading

Bar-Hillel A, Hertz T, Shental N, Weinshall D (2005) Learning a Mahalanobis metric from equivalence constraints. J Mach Learn Res 6:937–965

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

Basu S, Davidson I, Wagstaff K (eds) (2008) Constrained clustering: advances in algorithms, theory, and applications. CRC Press, Boca Raton

Bilenko M, Basu S, Mooney RJ (2004) Integrating constraints and metric learning in semi-supervised clustering. In: Proceedings of the twenty-first international conference on machine learning, Banff, pp 11–18

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc 39(1):1–38

Kamvar S, Klein D, Manning CD (2003) Spectral learning. In: Proceedings of the international joint conference on artificial intelligence, Acapulco, pp 561–566

Klein D, Kamvar SD, Manning CD (2002) From instance-level constraints to space-level constraints: making the most of prior knowledge in data clustering. In: Proceedings of the nineteenth international conference on machine learning, Sydney, pp 307–313

Lu Z, Leen T (2005) Semi-supervised learning with penalized probabilistic clustering. In: Advances in neural information processing systems, vol 17. MIT Press, Cambridge, MA, pp 849–856

MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth symposium on math, statistics, and probability, vol 1. University of California Press, California, pp 281–297

Shental N, Bar-Hillel A, Hertz T, Weinshall D (2004) Computing Gaussian mixture models with EM using equivalence constraints. In: Advances in neural information processing systems, vol 16. MIT Press, Cambridge, MA, pp 465–472

Wagstaff K, Cardie C (2000) Clustering with instance-level constraints. In: Proceedings of the seventeenth international conference on machine learning. Morgan Kaufmann, San Francisco, pp 1103–1110

Wagstaff K, Cardie C, Rogers S, Schroedl S (2001) Constrained k-means clustering with background knowledge. In: Proceedings of the eighteenth international conference on machine learning. Morgan Kaufmann, San Francisco, pp 577–584

Xing EP, Ng AY, Jordan MI, Russell S (2003) Distance metric learning, with application to clustering with side-information. In: Advances in neural information processing systems, vol 15. MIT Press, Cambridge, MA, pp 505–512

Constraint Classification

▶ Preference Learning

Constraint-Based Mining

Siegfried Nijssen
Katholieke Universiteit Leuven, Leuven, Belgium

Definition

Constraint-based mining is the research area studying the development of data mining algorithms that search through a pattern or model space restricted by constraints. The term is usually used to refer to algorithms that search for patterns only. The most well-known instance of constraint-based mining is the mining of ▶ frequent patterns. Constraints are needed in pattern mining algorithms to increase the efficiency of the search and to reduce the number of patterns that are presented to the user, thus making knowledge discovery more effective and useful.

Motivation and Background

Constraint-based pattern mining is a generalization of frequent itemset mining. For an introduction to frequent itemset mining, see ▶ Frequent Pattern. A constraint-based mining problem is specified by providing the following elements:

- A database $\mathcal{D}$, usually consisting of independent transactions (or instances)
- A ▶ hypothesis space $\mathcal{L}$ of patterns
- A constraint $q(\theta, \mathcal{D})$ expressing criteria that a pattern θ in the hypothesis space should fulfill on the database

The general constraint-based mining problem is to find the set

$$\text{Th}(\mathcal{D}, \mathcal{L}, q) = \{\theta \in \mathcal{L} | q(\theta, \mathcal{D}) = \text{true}\}.$$

Alternative problem settings are obtained by making different choices for $\mathcal{D}, \mathcal{L}$ and q. For instance,

- If the database and hypothesis space consist of itemsets, and the constraint checks if the support of a pattern exceeds a predefined threshold in data, the frequent itemset mining problem is obtained (see ▶ Frequent Pattern)
- If the database and the hypothesis space consist of graphs or trees instead of itemsets, a graph mining or a tree mining problem is obtained. For more information about these topics, see ▶ Graph Mining and ▶ Tree Mining
- Additional syntactic constraints can be imposed

An overview of important types of constraints is given below.

One can generalize the constraint-based mining problem beyond pattern mining. Also models, such as ▶ Decision Trees, could be seen as languages of interest. In the broadest sense, topics such as ▶ Constrained Clustering, ▶ Cost-Sensitive Learning, and even learning ▶ Support Vector Machines (SVMs) may be seen as constraint-based mining problems. However, it is currently not common to categorize these topics as *constraint-based mining*; in practice, the term refers to constraint-based *pattern* mining.

From the perspective of constraint-based mining, the knowledge discovery process can be seen as a process in which a user repeatedly specifies constraints for data mining algorithms; the data mining system is a solver that finds patterns or models that satisfy the constraints.

This approach to data mining is very similar to querying relational databases. Whereas relational databases are usually queried using operations such as projections, selections, and joins, in the constraint-based mining framework data is queried to find patterns or models that satisfy constraints that cannot be expressed in these primitives. A database which supports constraint-based mining queries, stores patterns and models, and allows later reuse of patterns and models, is sometimes also called an *inductive database* (Imielinski and Mannila 1996).

Structure of the Learning System

Constraints

Frequent pattern mining algorithms can be generalized along several dimensions.

One way to generalize pattern mining algorithms is to allow them to deal with arbitrary coverage relations, which determine when a pattern matches a transaction in the data. In the example of mining itemsets, the subset relation determines the coverage relation. The coverage relation is at the basis of constraints such as minimum support; an alternative coverage relation would be the superset relation.

From the coverage relation follows a generality relationship. A pattern θ_1 is defined to be more specific than a pattern θ_2 (denoted by $\theta_1 \succ \theta_2$) if any transaction that is covered by θ_1 is also covered by θ_2 (see ▶ Generalization). In frequent itemset mining, itemset I_1 is more general than itemset I_2 if and only $I_1 \subseteq I_2$.

Generalization and coverage relationships can be used to identify the following types of constraints.

Monotonic and Anti-Monotonic Constraints

An essential property which is exploited in ▶ frequent pattern mining, is that allsubsets of a frequent pattern are also frequent. This is a property that can begeneralized:

- A constraint is called *monotonic* if any generalization of a pattern that satisfies the constraint, also satisfies the constraint
- A constraint is called *anti-monotonic* if any specialization of a pattern that satisfies the constraint, also satisfies the constraint

In some publications, the definitions of monotonic and anti-monotonic are used reversely.

The following are examples of monotonic constraints:

- Minimum support
- Syntactic constraints, for instance: a constraint that requires that patterns specializing a given pattern x are excluded a constraint requiring patterns to be small given a definition of pattern size
- Disjunctions or conjunctions of monotonic constraints
- Negations of anti-monotonic constraints

The following are examples of anti-monotonic constraints:

- Maximum support
- Syntactic constraints, for instance, a constraint that requires that patterns generalizing a given pattern x are excluded
- Disjunctions or conjunctions of anti-monotonic constraints
- Negations of monotonic constraints

Succinct Constraints

Constraints that can be pushed in the mining process by adapting the pattern space or data, are called succinct constraints. An example of a succinct constraint is the monotonic constraint that an itemset should contain the item A. This constraint could be dealt with by deleting all transactions that do not contain A. For any frequent itemset found in the new dataset, it is now known that the item A can be added to it.

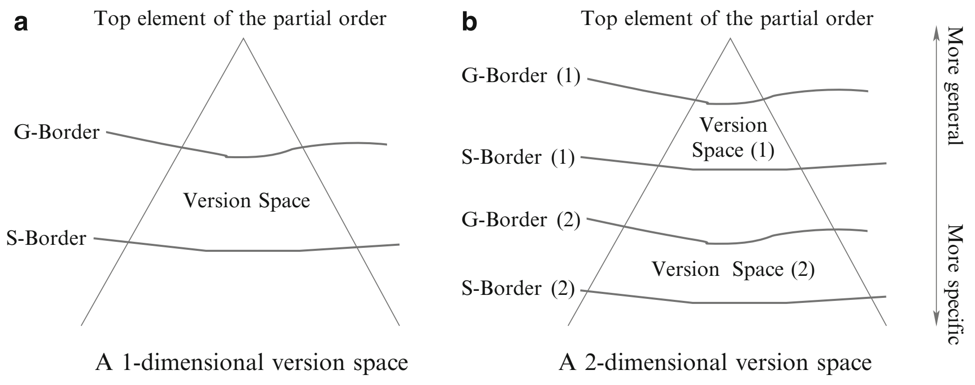
Convertible Constraints

Some constraints that are not monotonic, can still be *convertible* monotonic (Pei and Han 2002). A constraint is convertible monotonic if for every pattern θ one least general generalization θ' can be identified such that if θ satisfies the constraint, then θ' also satisfies the constraint. An example of a convertible constraint is a maximum average cost constraint. Assume that every item in an itemset has a cost as defined by a function $c(i)$. The constraint $c(I) = \sum_{i \in I} c(i)/|I| \leq maxcost$ is not monotonic. However, for every itemset I with $c(I) \leq maxcost$, if an item i is removed with $c(i) = \max_{i \in I} c(i)$, an itemset with $c(I - \{i\}) \leq c(I) \leq maxcost$ is obtained.

Maximum average cost has the desirable property that no access to the data is needed to identify the generalization that should satisfy the constraints. If it is not possible to identify the necessary least general generalization before accessing the data, the convertible constraint is also sometimes called weak (anti-)monotone (Zhu et al. 2007).

Boundable Constraints

Constraints on non-monotonic measures for which a monotonic bound exist, are called boundable. An example of such a constraint is a minimum accuracy constraint in a database with binary class labels. Assume that every itemset is interpreted as a rule **if** I **then** 1 **else** 2 (thus, class label 1 is predicted if a transaction contains itemset I, or class label 2 otherwise; see ▶ Supervised Descriptive Rule Induction). A minimum accuracy constraint can be formalized by the formula $(\text{fr}(I, D_1) + |D_2| - \text{fr}(I, D_2))/|D| \geq minacc$, where D_k is the



Constraint-Based Mining, Fig. 1 Version spaces

database containing only the examples labeled with class label k. It can be derived from this that

$$\mathrm{fr}(I, D_1) \geq |D| minacc - |D_2| +$$
$$\mathrm{fr}(I, D_2) \geq |D| minacc - |D_2|.$$

In other words, if a high accuracy is desirable, a minimum number of examples of class 1 is required to be covered, and a minimum frequency constraint can thus be derived. Therefore, minimum support can be used as a bound for minimum accuracy.

The principle of deriving bounds for non-monotonic measures can be applied widely (Bayardo et al. 1999; Morishita and Sese 2000).

Borders

If constraints are not restrictive enough, the number of patterns can be huge. Ignoring statistics about patterns such as their exact frequency, the set of patterns can be represented more compactly only by listing the patterns in the *border*(s) (Mannila and Toivonen 1997), similar to the idea of ▸ version spaces. An example of a border is the set of maximalfrequent itemsets (see ▸ Frequent Pattern). Borders can be computed for othertypes of both monotonic and anti-monotonic constraints as well. There areseveral complications compared to the simple frequent pattern miningsetting:

- If there is an anti-monotonic constraint, such as maximum support, not only is it needed to compute a border for the most specific elements in the set (S-Set), but also a border for the least general elements in the set (G-Set)
- If the formula is a disjunction of conjunctions, the result of a query becomes a union of version spaces, which is called a multi-dimensional version space (see Fig. 1) (De Raedt et al. 2002); the G-Set of one version space may be more general than the G-Set of another version space

Both the S-Set and the G-Set can be represented by listing elements just within the version space (the positive border), or elements just outside the version space (the negative border). For instance, the positive border of the G-Set consists of those patterns which are part of the version space, and for which no generalizations exist which are part of the version space.

Similarly, there may exist several representations of multi-dimensional version spaces; optimizing the representation of multi-dimensional version spaces is analogous to optimizing queries in relational databases (De Raedt et al. 2002).

Borders form a *condensed representations*, that is, they compactly represent the solution space; see ▸ Frequent Pattern.

Algorithms

For many of the constraints specified in the previous section specialized algorithms have been developed in combination with specific hypothesis spaces. It is beyond the scope of this chapter to discuss all these algorithms; only the most common ideas are provided here.

The main idea is that ▶ Apriori can easily be updated to deal with general monotonic constraints in arbitrary hypothesis spaces. The concept of a specialization refinement operator is essential to operateon other hypothesis spaces than itemsets. A specialization operator $\rho(\theta)$ computes a set of specializations in the hypothesis space for a given input pattern. In pattern mining, this operator should have the following properties:

- Completeness: every pattern in the hypothesis space should be reachable by repeated application of the refinement operator starting from the most general pattern in the hypothesis space
- Nonredundancy: every pattern in the hypothesis space should be reachable in only one way starting from the most general pattern in the hypothesis space

In itemset mining, optimal refinement is usually obtained by first ordering the items (for instance, alphabetically, or by frequency), and then adding items that are higher in the chosen order to a set than the items already in the set. For instance, for the itemset $\{A, C\}$, the specialization operator returns $\rho(\{A, C\}) = \{\{A, C, D\}, \{A, C, E\}\}$, assuming that the domain of items $\{A, B, C, D, E\}$ is considered. Other refinement operators are needed while dealing with other hypothesis spaces, such as in ▶ graph mining.

The search in Apriori proceeds breadth-first. Each level, the specialization operator is applied on patterns satisfying the monotonic constraints to generate candidates for the next level. For every new candidate it is checked whether its generalizations satisfy the monotonic constraints. To create a set of generalizations, a generalization refinement operator can be used. In frequent itemset mining, usually single items are removed from the itemset to generate generalizations.

More changes are required to deal with *anti-monotonic* constraints. A simple way of dealing with both monotonic and anti-monotonic constraints is to first compute all patterns that satisfy the monotonic constraints, and then to prune the patterns that fail to satisfy the anti-monotonic constraints. More challenging is to "push" anti-monotonic constraints in the mining process. An observation which is often exploited is that generalizations of patterns that do not satisfy the anti-monotonic constraint need not be considered. Well-known strategies are:

- In a breadth-first setting: traverse the lattice in reverse order for monotonic constraints, after the patterns have been determined satisfying the anti-monotonic constraints (De Raedt et al. 2002)
- In a depth-first setting: during the search for patterns, try to guess the largest pattern that can still be reached, and prune a branch in the search if the pattern does not satisfy the monotonic constraint on this pattern (Bucila et al. 2003; Kifer et al. 2003)

It is beyond the scope of this chapter to discuss how to deal with other types of constraints; however, it should be pointed out that not all combinations of constraints and hypothesis spaces have been studied; it is not obvious whether all constraints can be pushed usefully in a pattern search for any hypothesis space, for instance, when boundable constraints in more complex hypothesis spaces (such as graphs) are involved. Research in this area is ongoing.

Cross-References

▶ Constrained Clustering
▶ Frequent Pattern
▶ Graph Mining
▶ Tree Mining

Recommended Reading

Bayardo RJ Jr, Agrawal R, Gunopulos D (1999) Constraint-based rule mining in large, dense databases. In: Proceedings of the 15th international conference on data engineering (ICDE), Sydney, pp 188–197

Bucila C, Gehrke J, Kifer D, White WM (2003) DualMiner: a dual-pruning algorithm for itemsets with constraints. Data Min Knowl Discov 7(3):241–272

De Raedt L, Jaeger M, Lee SD, Mannila H (2002) A theory of inductive query answering (extended abstract). In: Proceedings of the second IEEE international conference on data mining (ICDM). IEEE Press, Los Alamitos, pp 123–130

Imielinski T, Mannila H (1996) A database perspective on knowledge discovery. Commun ACM 39:58–64

Kifer D, Gehrke J, Bucila C, White WM (2003) How to quickly find a witness. In: Proceedings of the twenty-second ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems. ACM Press, San Diego, pp 272–283

Mannila H, Toivonen H (1997) Levelwise search and borders of theories in knowledge discovery. Data Min Knowl Discov 1(3):241–258

Morishita S, Sese J (2000) Traversing itemset lattices with statistical metric pruning. In: Proceedings of the nineteenth ACM SIGACT-SIGMOD-SIGART symposium on database systems (PODS). ACM Press, San Diego, pp 226–236

Pei J, Han J (2002) Constrained frequent pattern mining: a pattern-growth view. SIGKDD Explor 4(1):31–39

Zhu F, Yan X, Han J, Yu PS (2007) gPrune: a constraint pushing framework for graph pattern mining. In: Proceedings of the sixth Pacific-Asia conference on knowledge discovery and data mining (PAKDD). Lecture notes in computer science, vol 4426. Springer, Berlin, pp 388–400

Constructive Induction

Constructive induction is any form of ▸ induction that generates new descriptors not present in the input data (Dietterich and Michalski 1983).

Recommended Reading

Dietterich TG, Michalski RS (1983) A comparative review of selected methods for learning from examples. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach, Tioga, pp 41–81

Content Match

▸ Text Mining for Advertising

Content-Based Filtering

Synonyms

Content-based recommending

Definition

Content-based filtering is prevalent in ▸ Information Retrieval, where the text and multimedia content of documents is used to select documents relevant to a user's query. In the context this refers to content-based recommenders, that provide recommendations by comparing representations of content describing an item to representations of content that interests a user.

Content-Based Recommending

▸ Content-Based Filtering

Context-Sensitive Learning

▸ Concept Drift

Contextual Advertising

▸ Text Mining for Advertising

Continual Learning

Synonyms

Life-long learning

Definition

A learning system that can continue adding new data without the need to ever stop or freeze the updating. Usually continual learning requires incremental and ▶ online learning as a component, but not every incremental learning system has the ability to achieve continual learning, i.e., the learning may deterioate after some time.

Cross-References

▶ Cumulative Learning

Continuous Attribute

A **continuous attribute** can assume all values on the number line within the value range. See ▶ Attribute and ▶ Measurement Scales.

Contrast Set Mining

Definition

Contrast set mining is an area of ▶ supervised descriptive rule induction. The contrast set mining problem is defined as finding contrast sets, which are conjunctions of attributes and values that differ meaningfully in their distributions across groups (Bay and Pazzani 2001). In this context, groups are the properties of interest.

Recommended Reading

Bay SD, Pazzani MJ (2001) Detecting group differences: mining contrast sets. Data Mining Knowl Discov 5(3):213–246

Cooperative Coevolution

▶ Compositional Coevolution

Co-reference Resolution

▶ Entity Resolution
▶ Record Linkage

Correlation Clustering

Anthony Wirth
The University of Melbourne, Melbourne, VLC, Australia

Synonyms

Clustering with advice; Clustering with constraints; Clustering with qualitative information; Clustering with side information

Definition

In its rawest form, *correlation clustering* is graph optimization problem. Consider a ▶ clustering C to be a mapping from the elements to be clustered, V, to the set $\{1, \ldots, |V|\}$, so that u and v are in the same cluster if and only if $C[u] = C[v]$. *Given* a collection of items in which each pair (u, v) has two weights w^+_{uv} and w^-_{uv}, we must *find* a clustering C that minimizes

$$\sum_{C[u]=C[v]} w^-_{uv} + \sum_{C[u]\neq C[v]} w^+_{uv}, \qquad (1)$$

or, equivalently, maximizes

$$\sum_{C[u]=C[v]} w^+_{uv} + \sum_{C[u]\neq C[v]} w^-_{uv}. \qquad (2)$$

Note that although w^+_{uv} and w^-_{uv} may be thought of as positive and negative evidence towards coassociation, the actual weights are nonnegative.

Motivation and Background

The notion of *clustering with advice*, that is nonmetric-driven relations between items, had been studied in other communities (Ferligoj and Batagelj 1982) prior to its appearance in theoretical computer science. Traditional clustering problems, such as k-median and k-center, assume that there is some type of distance measure (metric) on the data items, and often specify the number of clusters that should be formed. In the clustering with advice framework, however, the number of clusters to be built need not be specified in advance: it can be an outcome of the objective function. Furthermore, instead of, or in addition to, a distance function, we are given advice as to which pairs of items are similar. The two weights w^+_{uv} and w^-_{uv} correspond to external advice about whether the pair should be clustered together or separately. Bansal et al. (2002) introduced the problem to the theoretical computer science and machine-learning communities. They were motivated by database consistency problems, in which the same entity appeared in different forms in various databases. Given a collection of such records from multiple databases, the aim is to cluster together the records that appear to correspond to the same entity. From this viewpoint, the log odds ratio from some classifier,

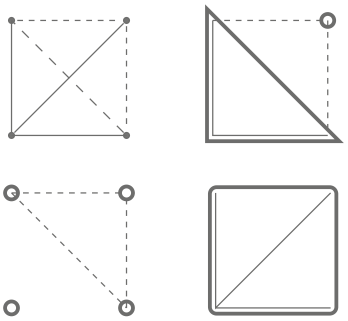
$$\log\left(\frac{\Pr(\text{same})}{\Pr(\text{different})}\right),$$

corresponds to a label w_{uv} for the pair. In many applications only one of the $+$ and $-$ weights for the pair is nonzero, that is

$$(w^+_{uv}, w^-_{uv}) = \begin{cases} (w_{uv}, 0) & \text{for } w_{uv} \geq 0 \\ (0, -w_{uv}) & \text{for } w_{uv} \leq 0. \end{cases}$$

In addition, if every pair has weight ± 1, then the instance is called *complete*, otherwise it is referred to as *general*. Demaine et al. (2006) suggest the following motivation. Suppose we have a set of guests at a party. Each guest has preferences for whom they would like to sit with, and for whom they would like to avoid. We must group the guests into tables in a way that enhances the amicability of the party.

The notion of producing good clusterings when given inconsistent advice first appeared in the work of Ben et al. (1999). A canonical example of inconsistent advice is this: items u and v are similar, items v and y are similar, but u and y are dissimilar. It is impossible to find a clustering that satisfies all the advice.



Correlation Clustering, Fig. 1 *Top left* is a toy *clustering with advice* example showing three similar pairs (*solid edges*) and three dissimilar pairs (*dashed edges*). *Bottom left* is a clustering solution for this example with four singleton clusters, while *bottom right* has one cluster. *Top right* is a partitioning into two clusters that appears to best respect the advice

Figure 1 shows a very simple example of inconsistent advice. In addition, although Correlation clustering is an NP-hard problem, recent algorithms for clustering with advice *guarantee* that their solutions are only a specified factor worse than the optimal: that is, they are *approximation algorithms*.

Theory

In setting out the correlation clustering framework, Bansal et al. (2002) noted that the following algorithm produces a 2-approximation for the maximization problem:

> If the total of the positive weights exceeds the total of the negative weights then, place all the items in a single cluster; otherwise, make each item a singleton cluster.

They then showed that complete instances are NP-hard to optimize, and how to minimize the penalty (1) with a constant factor approximation. The constant for this combinatorial algorithm was rather large. The algorithm relied heavily on the completeness of the instance; it iteratively *cleans* clusters until every cluster is δ-*clean*. That is, for each item at most a fraction $\delta (0 < \delta < 1)$ of the other items in its cluster have a negative relation with it, and at most δ outside its cluster a positive relation. Bansal et al. also demonstrated that the minimization problem on general instances is APX-hard: there is some constant, larger than 1, below which approximation is NP-hard. Finally, they provided a polynomial time approximation scheme (PTAS) for maximizing (2) in complete instances.

The constant factor for minimizing (1) on complete instances was improved to 4 by Charikar et al. (2003). They employed a region-growing type procedure to round the solution of a linear programming relaxation of the problem:

$$\begin{aligned}&\text{maximize}\\ &\qquad \sum_{ij} w_{ij}^{+} \cdot x_{ij} + w_{ij}^{-} \cdot (1 - x_{ij})\\ &\text{subject to}\\ &\qquad x_{ik} \leq x_{ij} + x_{jk} \quad \text{for all } i, j, k\\ &\qquad x_{ij} \in [0, 1] \qquad\quad \text{for all } i, j\end{aligned} \tag{3}$$

In this setting, $x_{ij} = 1$ implies i and j's separation, while $x_{ij} = 0$ implies coclustering, with values in between representing partial evidence. In practice solving this linear program is very slow and has huge memory demands (Bertolacci and Wirth 2007). Charikar et al. also showed that this version of problem is APX-hard.

For the maximization problem (2), they showed that instances with general weights were APX-hard and provided a rounding of the following semidefinite program (SDP) that yields a 0.7664 factor approximation algorithm.

$$\begin{aligned}&\text{maximize}\\ &\qquad \sum_{+(ij)} w_{ij}(v_i \cdot v_j) + \sum_{-(ij)} w_{ij}(1 - v_i \cdot v_j)\\ &\text{subject to}\\ &\qquad v_i \cdot v_i = 1 \quad \text{for all } i\\ &\qquad v_i \cdot v_j \geq 0 \quad \text{for all } i, j\end{aligned} \tag{4}$$

In this case we interpret $v_i \cdot v_j = 1$ as evidence that i and j are in the same cluster, but $v_i \cdot v_j = 0$ as evidence toward separation.

Emanuel and Fiat (2003) extended the work of Bansal et al. by drawing a link between Correlation Clustering and the Minimum Multicut problem. This reduction to Multicut provided an $O(\log n)$ approximation algorithm for minimizing general instances of Correlation Clustering. Interestingly, Emanuel and Fiat also showed that there was reduction in the opposite direction: an optimal solution to Correlation Clustering induced an optimal solution to Minimum Multicut.

Demaine and Immorlica (2003) also drew the link from Correlation Clustering to Minimum multicut and its $O(\log n)$ approximation algorithm. In addition, they described an $O(r^3)$-approximation algorithm for graphs that exclude the complete bipartite graph $K_{r,r}$ as a minor.

Swamy (2004), using the same SDP (4) as Charikar et al., but different rounding techniques, showed how to maximize (2) within factor 0.7666 in general instances.

The factor 4 approximation for minimization (1) of complete instances was lowered to 2.5 by Ailon et al. (2005). Using the *distances* obtained by solving the linear program (3), they repeat the following steps:

form a cluster around random item i by including each (unclustered) j with probability $1 - x_{ij}$; set the cluster aside.

Since solving the linear program is highly resource hungry, Ailon et al. provided a combinatorial alternative: add j to i's cluster if $w_{ij}^{+} > w_{ij}^{-}$. Not only is this algorithm very fast, it is actually a factor 3 approximation.

Recently, Tan (2007) has shown that the $79/80 + \epsilon$ inapproximability for maximizing (2) on general weighted graphs extends to general unweighted graphs.

A further variant in the Correlation Clustering family of problems is the maximization of (2)–(1), known as *maximizing correlation*. Charikar and Wirth (2004) proved an $\Omega(1/\log n)$ approximation for the general problem of maximizing

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j, \qquad \text{s.t } x_i \in \{-1, 1\} \text{ for all } i, \tag{5}$$

for a matrix A with null diagonal entries, by rounding the canonical SDP relaxation. This effectively maximized correlation with the requirement that two clusters be formed; it was not hard to extend this to general instances. The gap between the vector SDP solution and the integral solution to maximizing the quadratic program (5) was in fact shown to be $\Theta(1/\log n)$ in general (Alon et al. 2006). However, in other instances such as those with a bounded number of nonzero weights for each item, a constant factor approximation was possible. Arora et al. (2005) went further and showed that it is *quasi*-NP-hard to approximate the maximization to a factor better than $\Omega(1/\log^{\gamma} n)$ for some $\gamma > 0$.

Shamir et al. (2004) showed that ▸ Cluster Editing and p-Cluster Editing, in which p clusters must be formed, are NP-complete (for $p \geq 2$). Gramm et al. (2004) took an innovative approach to solving the Clustering Editing problem exactly. They had previously produced an $O(2.27k + n^3)$ time hand-made search tree algorithm, where k is the number of edges that need to be modified. This "awkward and error-prone work" was then replaced with a computer program that itself designed a search tree algorithm, involving automated case analysis, that ran in $O(1.92^k + n^3)$ time.

Kulis et al. (2005) unify various forms of clustering, correlation clustering, spectral clustering, and clustering with constraints in their kernel-based approach to k-means. In this, they have a general objective function that includes penalties for violating pairwise constraints and for having points spread far apart from their cluster centers, where the *spread* is measured in some high-dimensional space.

Applications

The work of Demaine and Immorlica (2003) on Correlation Clustering was closely linked with that of Bejerano et al. on Location Area Planning. This problem is concerned with the allocation of cells in a cellular network to clusters known as *location areas*. There are costs associated with traffic between the location areas (cuts between clusters) and with the size of clusters themselves (related to paging phones within individual cells). These costs drive the clustering solution in opposite directions, on top of which there are constraints on cells that must (or cannot) be in the same cluster. The authors show that the same $O(\log n)$ region-growing algorithm for minimizing Correlation Clustering and Multicut applies to Location Area Planning.

Correlation clustering has been directly applied to the coreference problem in natural language processing and other instances in which there are multiple references to the same object (Daume 2006; McCallum and Wellner 2005). Assuming some sort of undirected graphical model, such as a Conditional Random Field, algorithms for correlation clustering are used to partition a graph whose edge weights corresponding to log-potentials between node pairs. The machine learning community has applied some of the algorithms for Correlation clustering to problems such as email clustering and image segmentation. With similar applications in mind, Finley and Joachims (2005) explore the idea of adapting the pairwise input information to fit example clusterings given by a user. Their objective function is the same as Correlation Clustering (2), but their main tool is the ▸ Support Vector Machine.

There has been considerable interest in the ▸ consensus clustering problem, which is an excellent application of Correlation clustering techniques. Gionis et al. (2005) note several sources of motivation for the Consensus Clustering; these include identifying the correct number of clusters and improving clustering robustness. They adapt Charikar et al.'s region-growing algorithm to create a three-approximation that performs reasonably well in practice, though not as well as local search techniques. Gionis et al. also suggest using sampling as a tool for handling large data sets. Bertolacci and Wirth (2007) extended this study by implementing Ailon et al.'s algorithms with sampling, and therefore a variety of ways of developing a full clustering from the clustering of the sample. They noted that LP-based methods performed best, but placed a significant strain on resources.

Applications of Clustering with Advice

The ▸ *k*-means clustering algorithm is perhaps the most-used clustering technique: Wagstaff et al. incorporated constraints into a highly cited *k*-means variant called COP-KMEANS. They applied this algorithm to the task of identifying lanes of traffic based on input GPS data.

In the constrained-clustering framework, the constraints are usually assumed to be consistent (noncontradictory) and hard. In addition to the usual must- and cannot-link constraints, Davidson and Ravi (2005) added constraints enforcing various requirements on the distances between points in particular clusters. They analyzed the computational feasibility of the problem of establishing the (in) feasibility of a set of constraints, for various constraint types. Their constrained *k*-means algorithms were used to help a robot discover objects in a scene.

Recommended Reading

Ailon N, Charikar M, Newman A (2005) Aggregating inconsistent information: ranking and clustering. In: Proceedings of the thirty-seventh ACM symposium on the theory of computing. ACM Press, New York, pp 684–693

Alon N, Makarychev K, Makarychev Y, Naor A (2006) Quadratic forms on graphs. Invent Math 163(3):499–522

Arora S, Berger E, Hazan E, Kindler G, Safra S (2005) On non-approximability for quadratic programs. In: Proceedings of forty-sixth symposium on foundations of computer science. IEEE Computer Society, Washington, DC, pp 206–215

Bansal N, Blum A, Chawla S (2002) Correlation clustering. In: Correlation clustering. IEEE Computer Society, Washington, DC pp 238–247

Ben-Dor A, Shamir R, Yakhini Z (1999) Clustering gene expression patterns. J Comput Biol 6: 281–297

Bertolacci M, Wirth A (2007) Are approximation algorithms for consensus clustering worthwhile? In: Proceedings of seventh SIAM international conference on data mining. SIAM, Philadelphia, pp 437–442

Charikar M, Guruswami V, Wirth A (2003) Clustering with qualitative information. In: Proceedings of forty fourth FOCS, Cambridge, pp 524–533

Charikar M, Wirth A (2004) Maximizing quadratic programs: extending Grothendieck's inequality. In: Proceedings of forty fifth FOCS, Rome, pp 54–60

Daume H (2006) Practical structured learning techniques for natural language processing. PhD thesis, University of Southern California

Davidson I, Ravi S (2005) Clustering with constraints: feasibility issues and the *k*-means algorithm. In: Proceedings of fifth SIAM international conference on data mining, Newport Beach

Demaine E, Emanuel D, Fiat A, Immorlica N (2006) Correlation clustering in general weighted graphs. Theor Comput Sci 361(2):172–187

Demaine E, Immorlica N (2003) Correlation clustering with partial information. In: Proceedings of sixth workshop on approximation algorithms for combinatorial optimization problems, pp 1–13

Emanuel D, Fiat A (2003) Correlation clustering – minimizing disagreements on arbitrary weighted graphs. In: Proceedings of eleventh European symposium on algorithms, Budapest, pp 208–220

Ferligoj A, Batagelj V (1982) Clustering with relational constraint. Psychometrika 47(4):413–426

Finley T, Joachims T (2005) Supervised clustering with support vector machines. In: Proceedings of twenty-second international conference on machine learning, Bonn

Gionis A, Mannila H, Tsaparas P (2005) Clustering aggregation. In: Proceedings of twenty-first international conference on data engineering, Tokyo

Gramm J, Guo J, Hüffner F, Niedermeier R (2004) Automated generation of search tree algorithms for hard graph modification problems. Algorithmica 39(4):321–347

Kulis B, Basu S, Dhillon I, Mooney R (2005) Semi-supervised graph clustering: a kernel approach. In: Proceedings of twenty-second international conference on machine learning, Bonn, pp 457–464

McCallum A, Wellner B (2005) Conditional models of identity uncertainty with application to noun coreference. In: Saul L, Weiss Y, Bottou L (eds) Advances in neural information processing systems 17. MIT Press, Cambridge, pp 905–912

Meilă M (2003) Comparing clusterings by the variation of information. In: Proceedings of sixteenth conference on learning theory, pp 173–187

Shamir R, Sharan R, Tsur D (2004) Cluster graph modification problems. Discr Appl Math 144:173–182

Swamy C (2004) Correlation clustering: maximizing agreements via semidefinite programming. In: Proceedings of fifteenth ACM-SIAM symposium on discrete algorithms, pp 519–520

Tan J (2007) A note on the inapproximability of correlation clustering. Technical report 0704.2092, eprint arXiv, 2007

Correlation-Based Learning

▶ Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity

Cost

In ▶ Markov decision processes, negative *rewards* are often expressed as *costs*. A reward of $-x$ is expressed as a cost of x. In ▶ supervised learning, *cost* is used as a synonym for ▶ loss.

Cross-References

▶ Loss

Cost Function

▶ Loss Function

Cost-Sensitive Classification

▶ Cost-Sensitive Learning

Cost-Sensitive Learning

Charles X. Ling and Victor S. Sheng
The University of Western Ontario, London, ON, Canada

Synonyms

Cost-sensitive classification; Learning with different classification costs

Definition

Cost-Sensitive Learning is a type of learning that takes the misclassification costs (and possibly other types of cost) into consideration. The goal of this type of learning is to minimize the total cost. The key difference between cost-sensitive learning and cost-insensitive learning is that cost-sensitive learning treats different misclassifications differently. That is, the cost for labeling a positive example as negative can be different from the cost for labeling a negative example as positive. Cost-insensitive learning does not take misclassification costs into consideration.

Motivation and Background

Classification is an important task in inductive learning and machine learning. A classifier, trained from a set of training examples with class labels, can then be used to predict the class labels of new examples. The class label is usually discrete and finite. Many effective classification algorithms have been developed, such as ▶ naïve Bayes, ▶ decision trees, ▶ neural networks, and ▶ support vector machines. However, most classification algorithms seek to minimize the error rate: the percentage of the incorrect prediction of class labels. They ignore the difference between types of misclassification errors. In particular, they implicitly assume that all misclassification errors have equal cost.

In many real-world applications, this assumption is not true. The differences between different misclassification errors can be quite large. For example, in medical diagnosis of a certain cancer (where having cancer is regarded as the positive class, and non-cancer (healthy) as negative), misdiagnosing a cancer patient as healthy (the patient is actually positive but is classified as negative; thus it is also called "false negative") is much more serious (thus expensive) than a false-positive error. The patient could lose his/her life because of a delay in correct diagnosis and treatment. Similarly, if carrying a bomb is positive, then it is much more expensive to miss a terrorist who carries a bomb onto a flight than searching an innocent person.

Cost-sensitive learning takes costs, such as the misclassification cost, into consideration. Turney (2000) provides a comprehensive survey of a large variety of different types of costs in data

Cost-Sensitive Learning, Table 1 An example of cost matrix for binary classification

	Actual negative	Actual positive
Predict negative	$C(0,0)$, or *TP*	$C(0,1)$, or *FN*
Predict positive	$C(1,0)$, or FP	$C(1,1)$, or TP

mining and machine learning, including misclassification costs, data acquisition cost (instance costs and attribute costs), ▸ active learning costs, computation cost, human–computer interaction cost, and so on. The misclassification cost is singled out as the most important cost, and it has received the most attention in recent years.

Theory

The theory of cost-sensitive learning (Elkan 2001; Zadrozny and Elkan 2001) describes how the misclassification cost plays its essential role in various cost-sensitive learning algorithms.

Without loss of generality, binary classification is assumed (i.e., positive and negative class) in this paper. In cost-sensitive learning, the costs of false positive (actual negative but predicted as positive; denoted as *FP*), false negative (*FN*), true positive (*TP*), and true negative (*TN*) can be given in a cost matrix, as shown in Table 1. In the table, the notation $C(i,j)$ is also used to represent the misclassification cost of classifying an instance from its actual class j into the predicted class i (1 is used for positive, and 0 for negative). These misclassification cost values can be given by domain experts, or learned via other approaches. In cost-sensitive learning, it is usually assumed that such a cost matrix is given and known. For multiple classes, the cost matrix can be easily extended by adding more rows and more columns.

Note that $C(i,i)$ (*TP* and *TN*) is usually regarded as the "benefit" (i.e., negated cost) when an instance is predicted correctly. In addition, cost-sensitive learning is often used to deal with datasets with very imbalanced class distributions (see ▸ Class Imbalance Problem) (Japkowicz and Stephen 2002). Usually (and without loss of generality), the minority or rare class is regarded as the positive class, and it is often more expensive to misclassify an actual positive example into negative, than an actual negative example into positive. That is, the value of $FN = C(0,1)$ is usually larger than that of $FP = C(1,0)$. This is true for the cancer example mentioned earlier (cancer patients are usually rare in the population, but predicting an actual cancer patient as negative is usually very costly) and the bomb example (terrorists are rare).

Given the cost matrix, an example should be classified into the class that has the minimum expected cost. This is the minimum expected cost principle. The expected cost $R(i|x)$ of classifying an instance x into class i (by a classifier) can be expressed as:

$$R(i|x) = \sum_j P(j|x)C(j,i), \qquad (1)$$

where $P(j|x)$ is the probability estimation of classifying an instance into class j. That is, the classifier will classify an instance x into positive class if and only if:

$$P(0|x)C(1,0) + P(1|x)C(1,1) \le P(0|x)C(0,0) + P(1|x)C(0,1)$$

This is equivalent to:

$$P(0|x)C(1,0) - C(0,0) \le P(1|x)(C(0,1) - C(1,1))$$

Thus, the decision (of classifying an example into positive) will not be changed if a constant is added into a column of the original cost matrix. Thus, the original cost matrix can always be converted to a simpler one by subtracting $C(0,0)$ to the first column, and $C(1,1)$ to the second column. After such conversion, the simpler cost matrix is shown in Table 2. Thus, any given cost-matrix can be converted to one with $C(0,0) = C(1,1) = 0$. (Here it is assumed that the misclassification cost is the same for all examples. This property is a special case of the one discussed in Elkan (2001).) In the rest of the paper, it will be assumed that $C(0,0) = C(1,1) = 0$. Under this

Cost-Sensitive Learning, Table 2 A simpler cost matrix with an equivalent optimal classification

	True negative	True positive
Predict negative	0	$C(0,1) - C(1,1)$
Predict positive	$C(1,0) - C(0,0)$	0

assumption, the classifier will classify an instance x into positive class if and only if:

$$P(0|x)C(1,0) \leq P(1|x)C(0,1)$$

As $P(0|x) = 1 - P(1|x)$, a threshold p^* can be obtained for the classifier to classify an instance x into positive if $P(1|x) \geq p^*$, where

$$P^* = \frac{C(1,0)}{C(1,0) + C(0,1)}. \tag{2}$$

Thus, if a cost-insensitive classifier can produce a posterior probability estimation $p(1|x)$ for each test example x, one can make the classifier cost-sensitive by simply choosing the classification threshold according to (2), and classify any example to be positive whenever $P(1|x) \geq p^*$. This is what several cost-sensitive meta-learning algorithms, such as *Relabeling*, are based on (see later for details). However, some cost-insensitive classifiers, such as C4.5, may not be able to produce accurate probability estimation; they return a class label without a probability estimate. *Empirical Thresholding* (Sheng and Ling 2006) does not require accurate estimation of probabilities – an accurate ranking is sufficient. It simply uses ▸ cross-validation to search for the best probability value $p*$ to use as a threshold.

Traditional cost-insensitive classifiers are designed to predict the class in terms of a default, fixed threshold of 0.5. Elkan (2001) shows that one can "rebalance" the original training examples by sampling, such that the classifiers with the 0.5 threshold is equivalent to the classifiers with the p^* threshold as in (2), in order to achieve cost-sensitivity. The rebalance is done as follows. If all positive examples (as they are assumed as the rare class) are kept, then the number of negative examples should be multiplied by $C(1,0)/C(0,1) = FP/FN$. Note that as usualy $FP < FN$, the multiple is less than 1. This is, thus, often called "under-sampling the majority class." This is also equivalent to "proportional sampling," where positive and negative examples are sampled by the ratio of:

$$p(1)FN : p(0)FP \tag{3}$$

where $p(1)$ and $p(0)$ are the prior probability of the positive and negative examples in the original training set. That is, the prior probabilities and the costs are interchangeable: doubling $p(1)$ has the same effect as doubling *FN*, or halving *FP* (Drummond and Holte 2000). Most sampling meta-learning methods, such as costing (Zadrozny et al. 2003), are based on (3) above (see later for details).

Almost all meta-learning approaches are either based on (2) or (3) for the thresholding- and sampling-based meta-learning methods, respectively, to be discussed in the next section.

Structure of Learning System

Broadly speaking, cost-sensitive learning can be categorized into two categories. The first one is to design classifiers that are cost-sensitive in themselves. They are called the direct method. Examples of direct cost-sensitive learning are ICET (Turney 1995) and cost-sensitive decision tree (Drummond and Holte 2000; Ling et al. 2004). The other category is to design a "wrapper" that converts any existing cost-insensitive (or cost-blind) classifiers into cost-sensitive ones. The wrapper method is also called cost-sensitive meta-learning method, and it can be further categorized into thresholding and sampling. Here is a hierarchy of the cost-sensitive learning and some typical methods. This paper will focus on cost-sensitive meta-learning that considers the misclassification cost only.

Cost-Sensitive learning

– Direct methods
  - ICET (Turney 1995)
  - Cost-sensitive decision trees (Drummond and Holte 2000; Ling et al. 2004)

- Meta-learning
  - Thresholding
    - MetaCost (Domingos 1999)
    - CostSensitiveClassifier (CSC in short) (Witten and Frank 2005)
    - Cost-sensitive naïve Bayes (Chai et al. 2004)
    - Empirical Thresholding (ET in short) (Sheng and Ling 2006)
  - Sampling
    - Costing (Zadrozny et al. 2003)
    - Weighting (Ting 1998)

Direct Cost-Sensitive Learning

The main idea of building a direct cost-sensitive learning algorithm is to directly introduce and utilize misclassification costs into the learning algorithms. There are several works on direct cost-sensitive learning algorithms, such as ICET (Turney 1995) and cost-sensitive decision trees (Ling et al. 2004).

ICET (Turney 1995) incorporates misclassification costs in the fitness function of genetic algorithms. On the other hand, cost-sensitive decision tree (Ling et al. 2004), called CSTree here, uses the misclassification costs directly in its tree building process. That is, instead of minimizing entropy in attribute selection as in C4.5, CSTree selects the best attribute by the expected total cost reduction. That is, an attribute is selected as a root of the (sub) tree if it minimizes the total misclassification cost.

Note that as both ICET and CSTree directly take costs into model building, they can also take easily attribute costs (and perhaps other costs) directly into consideration, while meta cost-sensitive learning algorithms generally cannot.

Drummond and Holte (2000) investigate the cost-sensitivity of the four commonly used attribute selection criteria of decision tree learning: accuracy, Gini, entropy, and DKM. They claim that the sensitivity of cost is highest with the accuracy, followed by Gini, entropy, and DKM.

Cost-Sensitive Meta-Learning

Cost-sensitive meta-learning converts existing cost- insensitive classifiers into cost-sensitive ones without modifying them. Thus, it can be regarded as a middleware component that preprocesses the training data, or post-processes the output, from the cost-insensitive learning algorithms.

Cost-sensitive meta-learning can be further classified into two main categories: *thresholding* and *sampling*, based on (2) and (3) respectively, as discussed in the theory section.

Thresholding uses (2) as a threshold to classify examples into positive or negative if the cost-insensitive classifiers can produce probability estimations. *MetaCost* (Domingos 1999) is a *thresholding* method. It first uses bagging on decision trees to obtain reliable probability estimations of training examples, relabels the classes of training examples according to (2), and then uses the relabeled training instances to build a cost-insensitive classifier. *CSC* (Witten and Frank 2005) also uses (2) to predict the class of test instances. More specifically, *CSC* uses a cost-insensitive algorithm to obtain the probability estimations $P(j|x)$ of each test instance. (CSC is a meta-learning method and can be applied to any classifiers.) Then it uses (2) to predict the class label of the test examples. Cost-sensitive naïve Bayes (Chai et al. 2004) uses (2) to classify test examples based on the posterior probability produced by the naïve Bayes.

As seen, all *thresholding*-based meta-learning methods rely on accurate probability estimations of $p(1|x)$ for the test example x. To achieve this, Zadrozny and Elkan (2001) propose several methods to improve the calibration of probability estimates. *ET* (Empirical Thresholding) (Sheng and Ling 2006) is a thresholding-based meta-learning method. It does not require accurate estimation of probabilities – an accurate ranking is sufficient. *ET* simply uses cross-validation to search the best probability from the training instances as the threshold, and uses the searched threshold to predict the class label of test instances.

On the other hand, *sampling* first modifies the class distribution of the training data according to (3), and then applies cost-insensitive classifiers on the sampled data directly. There is no need for the classifiers to produce probability estimations, as long as they can classify positive or negative examples accurately. Zadrozny et al. (2003) show that proportional sampling with replacement produces duplicated cases in the training, which in turn produces overfitting in model building. Instead, Zadrozny et al. (2003) proposes to use "rejection sampling" to avoid duplication. More specifically, each instance in the original training set is drawn once, and accepted into the sample with the accepting probability $C(j,i)/Z$, where $C(j,i)$ is the misclassification cost of class i, and Z is an arbitrary constant such that $Z \geq \max C(j,i)$. When $Z = \max_{ij} C(j,i)$, this is equivalent to keeping all examples of the rare class, and sampling the majority class without replacement according to $C(1,0)/C(0,1)$ – in accordance with (3). Bagging is applied after rejection sampling to improve the results further. The resulting method is called *Costing*.

Weighting (Ting 1998) can also be viewed as a sampling method. It assigns a normalized weight to each instance according to the misclassification costs specified in (3). That is, examples of the rare class (which carries a higher misclassification cost) are assigned, proportionally, high weights. Examples with high weights can be viewed as example duplication – thus oversampling. *Weighting* then induces cost-sensitivity by integrating the instances' weights directly into C4.5, as C4.5 can take example weights directly in the entropy calculation. It works whenever the original cost-insensitive classifiers can accept example weights directly. (Thus, it can be said that *Weighting* is a semi meta-learning method.) In addition, *Weighting* does not rely on bagging as *Costing* does, as it "utilizes" all examples in the training set.

Recommended Reading

Chai X, Deng L, Yang Q, Ling CX (2004) Test-cost sensitive naïve Bayesian classification. In: Proceedings of the fourth IEEE international conference on data mining. IEEE Computer Society Press, Brighton

Domingos P (1999) MetaCost: a general method for making classifiers cost-sensitive. In: Proceedings of the fifth international conference on knowledge discovery and data mining, San Diego. ACM, New York, pp 155–164

Drummond C, Holte R (2000) Exploiting the cost (in)sensitivity of decision tree splitting criteria. In: Proceedings of the 17th international conference on machine learning, Stanford, pp 239–246

Elkan C (2001) The foundations of cost-sensitive learning. In: Proceedings of the 17th international joint conference of artificial intelligence. Morgan Kaufmann, Seattle, pp 973–978

Japkowicz N, Stephen S (2002) The class imbalance problem: a systematic study. Intell Data Anal 6(5):429–450

Ling CX, Yang Q, Wang J, Zhang S (2004) Decision trees with minimal costs. In: Proceedings of 2004 international conference on machine learning (ICML'2004), Banff

Sheng VS, Ling CX (2006) Thresholding for making classifiers cost-sensitive. In: Proceedings of the 21st national conference on artificial intelligence, 16–20 July 2006, Boston, pp 476–481

Ting KM (1998) Inducing cost-sensitive trees via instance weighting. In: Proceedings of the second European symposium on principles of data mining and knowledge discovery. Springer, Heidelberg, pp 23–26

Turney PD (1995) Cost-sensitive classification: empirical evaluation of a hybrid genetic decision tree induction algorithm. J Artif Intell Res 2:369–409

Turney PD (2000) Types of cost in inductive concept learning. In: Proceedings of the workshop on cost-sensitive learning at the 17th international conference on machine learning, Stanford University, Stanford

Witten IH, Frank E (2005) Data mining – practical machine learning tools and techniques with Java implementations. Morgan Kaufmann, San Francisco

Zadrozny B, Elkan C (2001) Learning and making decisions when costs and probabilities are both unknown. In: Proceedings of the seventh international conference on knowledge discovery and data mining, pp 204–213

Zadrozny B, Langford J, Abe N (2003) Cost-sensitive learning by cost-proportionate instance weighting. In: Proceedings of the third international conference on data mining, Melbourne

Cost-to-Go Function Approximation

▶ Value Function Approximation

Co-training

▶ Semi-supervised Learning

Covariance Matrix

Xinhua Zhang
NICTA, Australian National University, Canberra, ACT, Australia
School of Computer Science, Australian National University, Canberra, ACT, Australia
NICTA London Circuit, Canberra, ACT, Australia

Abstract
Covariance matrix is a generalization of covariance between two univariate random variables. It is composed of the pairwise covariance between components of a multivariate random variable. It underpins important stochastic processes such as Gaussian process, and in practice it provides key characterizations between multiple random factors.

Definition

It is convenient to define a covariance matrix by using multivariate random variables (*mrv*): $\mathbf{X} = (X_1, \ldots, X_d)^\top$. For univariate random variables X_i and X_j, their covariance is defined as

$$\mathrm{Cov}(X_i, X_j) = \mathbb{E}\left[(X_i - \mu_i)\left(X_j - \mu_j\right)\right],$$

where μ_i is the mean of X_i: $\mu_i = \mathbb{E}[X_i]$. As a special case, when $i = j$, then we get the variance of X_i, $\mathrm{Var}(X_i) = \mathrm{Cov}(X_i, X_i)$. Now in the setting of *mrv*, assuming that each component random variable X_i has finite variance under its marginal distribution, the covariance matrix $\mathrm{Cov}(\mathbf{X}, \mathbf{X})$ can be defined as a d-by-d matrix whose (i, j)th entry is the covariance:

$$\begin{aligned}(\mathrm{Cov}(\mathbf{X}, \mathbf{X}))_{ij} &= \mathrm{Cov}\left(X_i, X_j\right) \\ &= \mathbb{E}\left[(X_i - \mu_i)\left(X_j - \mu_j\right)\right].\end{aligned}$$

And its inverse is also called precision matrix.

It is easy to rewrite the element-wise definition into the matrix form:

$$\mathrm{Cov}(\mathbf{X}, \mathbf{X}) = \mathbb{E}\left[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^\top\right], \tag{1}$$

which naturally generalizes the variance of univariate random variables: $\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$.

Moreover, it is also straightforward to extend the covariance of a single *mrv* $\mathbf{X}$ to two *mrv*'s $\mathbf{X}$ (d dimensional) and $\mathbf{Y}$ (s dimensional), under the name cross covariance. It quantifies how much the component random variables in $\mathbf{X}$ and $\mathbf{Y}$ change together. The cross-covariance matrix is defined as a $d \times s$ matrix $\mathrm{Cov}(\mathbf{X}, \mathbf{Y})$ whose (i, j)th entry is

$$(\mathrm{Cov}(\mathbf{X}, \mathbf{Y}))_{ij} = \mathrm{Cov}(X_i, Y_j) \tag{2}$$

$$= \mathbb{E}\left[(X_i - \mathbb{E}[X_i])\left(Y_j - \mathbb{E}[Y_j]\right)\right]. \tag{3}$$

$\mathrm{Cov}(\mathbf{X}, \mathbf{Y})$ can also be written in the matrix form as

$$\mathrm{Cov}(\mathbf{X}, \mathbf{Y}) = \mathbb{E}\left[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{Y} - \mathbb{E}[\mathbf{Y}])^\top\right],$$

where the expectation is with respect to the joint distribution of $(\mathbf{X}, \mathbf{Y})$. Obviously, $\mathrm{Cov}(\mathbf{X}, \mathbf{Y})$ becomes $\mathrm{Cov}(\mathbf{X}, \mathbf{X})$ when $\mathbf{Y} = \mathbf{X}$.

Motivation and Background

The covariance between two univariate random variables measures how much they change together, and as a special case, the covariance of a random variable with itself is exactly its variance. It is important to note that covariance is an unnormalized measure of the correlation between the random variables.

As a generalization to multivariate random variables $\mathbf{X} = (X_1, \ldots, X_d)^\top$, the covariance matrix is a d-by-d matrix whose (i, j)th component is the covariance between X_i and X_j.

In many applications, it is important to characterize the relations between a set of factors, hence the covariance matrix plays an important role in practice, especially in machine learning.

Theory

Properties

Covariance $\mathrm{Cov}(\mathbf{X}, \mathbf{X})$ has the following properties:

1. Positive semi-definiteness. It follows from Eq. (1) that $\mathrm{Cov}(\mathbf{X}, \mathbf{X})$ is positive semi-definite. $\mathrm{Cov}(\mathbf{X}, \mathbf{X}) = \mathbf{0}$ if, and only if, $\mathbf{X}$ is constant almost surely, i.e., there exists a constant $\boldsymbol{x}$ such that $\Pr(\mathbf{X} \neq \boldsymbol{x}) = 0$. $\mathrm{Cov}(\mathbf{X}, \mathbf{X})$ is not positive definite if, and only if, there exists a constant $\boldsymbol{\alpha}$ such that $\langle \boldsymbol{\alpha}, \mathbf{X} \rangle$ is constant almost surely.
2. Relating cumulant to moments: $\mathrm{Cov}(\mathbf{X}, \mathbf{X}) = \mathbb{E}[\mathbf{X}\mathbf{X}^\top] - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{X}]^\top$.
3. Linear transform: If $\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{s \times d}$ and $\mathbf{b} \in \mathbb{R}^s$, then $\mathrm{Cov}(\mathbf{Y}, \mathbf{Y}) = \mathbf{A}\mathrm{Cov}(\mathbf{X}, \mathbf{X})\mathbf{A}^\top$.

Cross-covariance $\mathrm{Cov}(\mathbf{X}, \mathbf{Y})$ has the following properties:

1. Symmetry: $\mathrm{Cov}(\mathbf{X}, \mathbf{Y}) = \mathrm{Cov}(\mathbf{Y}, \mathbf{X})$.
2. Linearity: $\mathrm{Cov}(\mathbf{X}_1 + \mathbf{X}_2, \mathbf{Y}) = \mathrm{Cov}(\mathbf{X}_1, \mathbf{Y}) + \mathrm{Cov}(\mathbf{X}_2, \mathbf{Y})$.
3. Relating to covariance: If $\mathbf{X}$ and $\mathbf{Y}$ have the same dimension, then $\mathrm{Cov}(\mathbf{X} + \mathbf{Y}, \mathbf{X} + \mathbf{Y}) = \mathrm{Cov}(\mathbf{X}, \mathbf{X}) + \mathrm{Cov}(\mathbf{Y}, \mathbf{Y}) + 2\mathrm{Cov}(\mathbf{Y}, \mathbf{X})$.
4. Linear transform: $\mathrm{Cov}(\mathbf{A}\mathbf{X}, \mathbf{B}\mathbf{Y}) = \mathbf{A}\mathrm{Cov}(\mathbf{X}, \mathbf{Y})\mathbf{B}$.

It is highly important to note that $\mathrm{Cov}(\mathbf{X}, \mathbf{Y}) = 0$ is a necessary but not sufficient condition for $\mathbf{X}$ and $\mathbf{Y}$ to be independent.

Correlation Coefficient

Entries in the covariance matrix are sometimes presented in a normalized form by dividing each entry by its corresponding standard deviations. This quantity is called the correlation coefficient, represented as ρ_{X_i, X_j}, and defined as

$$\rho_{X_i, X_j} = \frac{\mathrm{Cov}(X_i, X_j)}{\mathrm{Cov}(X_i, X_i)^{1/2}\mathrm{Cov}(X_j, X_j)^{1/2}}.$$

The corresponding matrix is called the correlation matrix, and for Γ_X set to $\mathrm{Cov}(\mathbf{X}, \mathbf{X})$ with all non-diagonal entries zeroed, and Γ_Y likewise, then the correlation matrix is given by

$$\mathrm{Corr}(\mathbf{X}, \mathbf{Y}) = \Gamma_X^{-1/2}\mathrm{Cov}(\mathbf{X}, \mathbf{Y})\Gamma_Y^{-1/2}.$$

The correlation coefficient takes on values between $[-1, 1]$.

Parameter Estimation

Given observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ of a *mrv* $\mathbf{X}$, an unbiased estimator of $\mathrm{Cov}(\mathbf{X}, \mathbf{X})$ is

$$S = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^\top,$$

where $\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i$. The denominator $n-1$ reflects the fact that the mean is unknown and the sample mean is used in place. Note the maximum likelihood estimator in this case replaces the denominator $n-1$ by n.

Conjugate Priors

Covariance matrix is used to define the Gaussian distribution. In this case, the inverse Wishart distribution is the conjugate prior for the covariance matrix. Since the gamma distribution is a 1-D version of the Wishart distribution, hence in the 1-D case, the gamma is the conjugate prior for precision matrix.

Applications

Several key uses of the covariance matrix are reviewed here.

Correlation and Least Squares Approximation

In many machine learning problems, we often need to quantify the correlation of two *mrv*s which may be from two different spaces. For example, we may want to study how much the image stream of a movie is correlated with the comments it receives. For simplicity, we consider a r-dimensional *mrv* $\mathbf{X}$ and a s-dimensional *mrv* $\mathbf{Y}$. To study their correlation, suppose we have n pairs of observations $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$ drawn *iid* from certain underlying joint distribution of $(\mathbf{X}, \mathbf{Y})$. Let $\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i$ and $\bar{\boldsymbol{y}} = \frac{1}{n}\sum_{i=1}^n \boldsymbol{y}_i$ and stack $\{\boldsymbol{x}_i\}$ and $\{\boldsymbol{y}_i\}$ into $\tilde{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^\top$ and $\tilde{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)^\top$, respectively. Then the cross-covariance matrix $\mathrm{Cov}(\mathbf{X}, \mathbf{Y})$ can be estimated by $\frac{1}{n}\sum_{i=1}^n (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{y}_i - \bar{\boldsymbol{y}})^\top$. To quantify the cross correlation by a real number, we need to apply some norm of the cross-covariance matrix, and the simplest one is the Frobenius norm: $\|A\|_F^2 = \sum_{ij} A_{ij}^2$. Therefore we obtain a measure of cross correlation:

$$\left\| \frac{1}{n}\sum_{i=1}^n (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{y}_i - \bar{\boldsymbol{y}})^\top \right\|_F^2 = \frac{1}{n} H\tilde{X}\tilde{X}^\top H\tilde{Y}\tilde{Y}^\top, \tag{4}$$

where $H_{ij} = \delta_{ij} - \frac{1}{n}$ and $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. It is important to notice that (a) in this measure, inner product is performed only in the space of $\mathbf{X}$ and $\mathbf{Y}$ separately, i.e., no transformation between $\mathbf{X}$ and $\mathbf{Y}$ is required, and (b) the data points affect the measure only via inner products $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ as the (i, j)th entry of $\tilde{X}\tilde{X}^\top$ (and similarly for $\boldsymbol{y}_i$). Hence we can endow new inner products on $\mathbf{X}$ and $\mathbf{Y}$, which eventually allows us to apply kernels, e.g., Gretton et al. (2005). In a nutshell, kernels (Schölkopf and Smola 2002) redefine the inner product $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ by mapping $\boldsymbol{x}_i$ to a richer feature space via $\phi(\boldsymbol{x}_i)$ and then compute the inner product there: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) := \phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j)$. Since the measure in Eq. (4) only needs inner products, one can even directly define k without explicitly specifying ϕ. This allows us to (a) implicitly use a rich feature space whose dimension can be infinitely high and (b) apply this measure of independence to non-Euclidean spaces as long as a kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ can be defined on it.

Interestingly, this measure can be equivalently motivated by least square linear regression. That is, we look for a linear transform $T : \mathbb{R}^d \to \mathbb{R}^s$ which minimizes

$$\frac{1}{n}\sum_{i=1}^n \|(\boldsymbol{y}_i - \bar{\boldsymbol{y}}) - T(\boldsymbol{x}_i - \bar{\boldsymbol{x}})\|^2 .$$

And one can show that its minimum objective value is exactly equal to Eq. (4) up to a constant, as long as all $\boldsymbol{y}_i - \bar{\boldsymbol{y}}$ and $\boldsymbol{x}_i - \bar{\boldsymbol{x}}$ have unit length. In practice, this can be achieved by normalization. Or, the measure in Eq. (4) itself can be normalized by replacing the covariance matrix with the correlation matrix.

Principal Component Analysis

The covariance matrix plays a key role in principal component analysis (PCA). Assume we are given n *iid* observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ of a *mrv* $\mathbf{X}$, and let $\bar{\boldsymbol{x}} = \frac{1}{n}\sum_i \boldsymbol{x}_i$. PCA tries to find a set of orthogonal directions $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots$, such that the projection of $\mathbf{X}$ to the direction $\boldsymbol{w}_1$, $\boldsymbol{w}_1^\top \mathbf{X}$ has the highest variance among all possible directions in the d-dimensional space. After subtracting from $\mathbf{X}$ the projection to $\boldsymbol{w}_1$, $\boldsymbol{w}_2$ is chosen as the highest variance projection direction for the remainder. This procedure goes on, giving $\boldsymbol{w}_3, \ldots, \boldsymbol{w}_d$.

To find $\boldsymbol{w}_1 := \mathrm{argmax}_{\boldsymbol{w}} \mathrm{Var}(\boldsymbol{w}^\top \mathbf{X})$, we need an empirical estimate of $\mathrm{Var}(\boldsymbol{w}^\top \mathbf{X})$. Estimating $\mathbb{E}[(\boldsymbol{w}^\top \mathbf{X})^2]$ by $\boldsymbol{w}^\top \left(\frac{1}{n}\sum_i \boldsymbol{x}_i \boldsymbol{x}_i^\top\right) \boldsymbol{w}$ and $\mathbb{E}[\boldsymbol{w}^\top \mathbf{X}]$ by $\frac{1}{n}\sum_i \boldsymbol{w}^\top \boldsymbol{x}_i$, we get

$$\boldsymbol{w}_1 = \mathrm{argmax}_{\boldsymbol{w}:\|\boldsymbol{w}_1\|=1} \boldsymbol{w}^\top S \boldsymbol{w}, \quad \text{where}$$

$$S = \frac{1}{n}\sum_{i=1}^n (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^\top,$$

i.e., S is $\frac{n}{n-1}$ times the unbias empirical estimate of the covariance of $\mathbf{X}$, based on samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. $\boldsymbol{w}_1$ turns out to be exactly the eigenvector of S corresponding to the greatest eigenvalue.

Note that PCA is independent of the distribution of $\mathbf{X}$. More details on PCA can be found at Jolliffe (2002).

Gaussian Processes

Gaussian processes are another important framework in machine learning that relies on the covariance matrix. It is a distribution over functions $f(\cdot)$ from certain space $\mathcal{X}$ to $\mathbb{R}$, such that for any $n \in \mathbb{N}$ and any n points $\{\boldsymbol{x}_i \in \mathcal{X}\}_{i=1}^n$, the set of values of f evaluated at $\{\boldsymbol{x}_i\}_i$, $\{f(x_1), \ldots, f(x_n)\}$, will have an n-dimensional Gaussian distribution. Different choices of the covariance matrix of the multivariate Gaussian lead to different stochastic processes such as

Wiener process, Brownian motion, Ornstein-Uhlenbeck process, etc. In these cases, it makes more sense to define a covariance function $C : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, such that given any set $\{\boldsymbol{x}_i \in \mathcal{X}\}_{i=1}^n$ for any $n \in \mathbb{N}$, the n-by-n matrix $\left(C(\boldsymbol{x}_i, \boldsymbol{x}_j)\right)_{ij}$ is positive semi-definite and can be used as the covariance matrix. This further allows straightforward kernelization of a Gaussian process by using the kernel function as the covariance function.

Although the space of functions is infinite dimensional, the marginalization property of multivariate Gaussian distributions guarantees that the user of the model only needs to consider the observed $\boldsymbol{x}_i$ and ignore all the other possible $\boldsymbol{x} \in \mathcal{X}$. This important property says that for a *mrv* $\mathbf{X} = (\mathbf{X}_1^\top, \mathbf{X}_2^\top)^\top \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, the marginal distribution of $\mathbf{X}_1$ is $\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_{11})$, where Σ_{11} is the submatrix of Σ corresponding to $\mathbf{X}_1$ (and similarly for $\boldsymbol{\mu}_1$). So taking into account the random variable $\mathbf{X}_2$ will not change the marginal distribution of $\mathbf{X}_1$.

Cross-References

▶ Gaussian Distribution
▶ Gaussian Processes

Recommended Reading

For a complete treatment of covariance matrix from a statistical perspective, see Casella and Berger (2002) and Mardia et al. (1979) provides details for the multivariate case. PCA is comprehensively discussed in Jolliffe (2002), and kernel methods are introduced in Schölkopf and Smola (2002). Williams and Rasmussen (2006) gives the state of the art on how Gaussian processes can be utilized for machine learning.

Casella G, Berger R 2002 Statistical inference, 2nd edn. Duxbury, Pacific Grove

Gaussian Processes for Machine Learning, Carl Edward Rasmussen and Chris Williams, the MIT Press, Cambridge, MA, 2006

Gretton A, Herbrich R, Smola A, Bousquet O, Schölkopf B (2005) Kernel methods for measuring independence. J Mach Learn Res 6:2075–2129

Jolliffe IT (2002) Principal component analysis. Springer series in statistics, 2nd edn. Springer, New York

Mardia KV, Kent JT, Bibby JM (1979) Multivariate analysis. Academic, London/New York

Schölkopf B, Smola A (2002) Learning with Kernels. MIT, Cambridge

C

Covering Algorithm

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

Abstract

The covering algorithm is the dominant approach to classification ▶ rule learning. Its distinguishing feature is the idea to learn one rule at a time, successively removing all training examples that are covered by the learned rules.

Synonyms

Separate-and-conquer learning

Method

Most covering algorithms operate in a ▶ concept learning framework, i.e., they assume a set of positive and negative training examples. Adaptations to the multi-class case are typically performed via ▶ class binarization, learning different ▶ rule sets for binary problems. Some algorithms, most notably CN2 (Clark and Niblett 1989; Clark and Boswell 1991), learn multi-class rules directly by optimizing overall possible classes in the head of the rule. In this case, the resulting theory is interpreted as a ▶ decision list. In the following, we will assume a two-class problem with a positive and a negative class.

The COVERING algorithm starts with an empty theory. If there are any positive examples in the training set, it calls the subroutine FINDBESTRULE for learning a single rule that

```
procedure COVERING(Examples, Classifier)

Input: Examples, a set of positive and negative examples
for a class $c$.

// initialize the classifier
$Classifier = \emptyset$

//loop until no more positive examples are covered
while POSITIVE(Examples) $\neq \emptyset$ do

    // find the best rule for the current examples
    Rule = FINDBESTRULE(Examples)

    // check if we need more rules
    if RULESTOPPINGCRITERION(Classifier, Rule,
      Examples)
    then  break while

    // remove covered examples and add rule to rule set
    Examples = Examples \ COVER(Rule, Examples)
    $Classifier = Classifier \cup Rule$
endwhile

// post-process the rule set (e.g., pruning)
Classifier = POSTPROCESSING (Classifier)

Output: Classifier
```

will cover a subset of the positive examples (and possibly some negative examples as well). All covered examples are then separated from the training set, the learned rule is added to the theory, and another rule is learned from the remaining examples. Rules are learned in this way until no positive examples are left or until the RULESTOPPINGCRITERION fires. In the simplest case, the stopping criterion is a check whether there are still remaining positive examples that need to be covered. The resulting theory may also undergo some POSTPROCESSING, e.g., a separate pruning and re-induction phase as in RIPPER (Cohen 1995).

A more extensive survey of this family of algorithms can be found in Fürnkranz (1999a).

Cross-References

▶ Class Binarization
▶ Concept Learning
▶ Decision List
▶ Rule Learning
▶ Rule Set

Recommended Reading

Clark P, Boswell R (1991) Rule induction with CN2: some recent improvements. In: Proceedings of the 5th European working session on learning (EWSL-91), Porto. Springer, pp 151–163

Clark P, Niblett T (1989) The CN2 induction algorithm. Mach Learn 3(4):261–283

Cohen WW (1995) Fast effective rule induction. In: Prieditis A, Russell S (eds) Proceedings of the 12th international conference on machine learning (ML-95), Lake Tahoe. Morgan Kaufmann, pp 115–123

Fürnkranz J (1999) Separate-and-conquer rule learning. Artif Intell Rev 13(1):3–54. http://www.ofai.at/cgi-bin/tr-online?number+96-25

Credit Assignment

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

Synonyms

Structural credit assignment; Temporal credit assignment

Definition

When a learning system employs a complex decision process, it must assign credit or blame for the outcomes to each of its decisions. Where it is not possible to directly attribute an individual outcome to each decision, it is necessary to apportion credit and blame between each of the combinations of decisions that contributed to the outcome. We distinguish two cases in the credit assignment problem. *Temporal credit assignment* refers to the assignment of credit for outcomes to actions. *Structural credit assignment* refers to the assignment of credit for actions to internal decisions. The first subproblem involves determining when the actions that deserve credit were taken and the second involves assigning credit to the internal structure of actions (Sutton 1984).

Motivation

Consider the problem of learning to balance a pole that is hinged on a cart (Michie and Chambers 1968; Anderson and Miller 1991). The cart is constrained to run along a track of finite length and a fixed force can be applied to push the cart left or right. A controller for the pole and cart system must make a decision whether to push left or right at frequent, regular time intervals, for example, 20 times a second. Suppose that this controller is capable of learning from trial-and-error. If the pole falls over, then it must determine which actions it took helped or hurt its performance. Determining that action is the problem of *temporal credit assignment*. Although the actions are directly responsible for the outcome of a trial, the internal process for choosing the action indirectly affects the outcome. Assigning credit or blame to those internal processes that lead to the choice of action is the *structural credit assignment* problem. In the case of pole balancing, the learning system will typically keep statistics such as how long, on average, the pole remained balanced after taking a particular action in a particular state, or after a failure, it may count back and determine the average amount of time to failure after taking a particular action in a particular state. Using these statistics, the learner attempts to determine the best action for a given state.

The above example is typical of many problems in ▸ reinforcement learning (Sutton and Barto 1998), where an agent interacts with its environment and through that interaction, learns to improve its performance in a task. Although Samuel (1959) was the first to use a form of reinforcement learning in his checkers playing program, Minsky (1961) first articulated the credit assignment, as follows:

> Using devices that also learn which events are associated with reinforcement, i.e., reward, we can build more autonomous "secondary reinforcement" systems. In applying such methods to complex problems, one encounters a serious difficulty – in distributing credit for success of a complex strategy among the many decisions that were involved.

The BOXES algorithm of Michie and Chambers (1968) learned to control a pole balancer and performed credit assignment but the problem of credit assignment later became central to reinforcement learning, particularly following the work of Sutton (1984). Although credit assignment has become most strongly identified with reinforcement learning, it may appear in any learning system that attempts to assess and revise its decision-making process.

Structural Credit Assignment

The setting for our learning system is that we have an agent that interacts with an environment. The environment may be a virtual one, as in game playing, or it may be physical, as in a robot performing some task. The agent receives input, possibly through sensing devices, that allows it to characterize the state of the world. Somehow, the agent must map these inputs to appropriate responses. These responses may change the state of the world. In reinforcement learning, we assume that the agent will receive some reward signal after an action or sequence of actions. Its job is to maximize these rewards over time.

Structural credit assignment is associated with generalization over the input space of the agent. For example, a game player may have to respond to a very large number of potential board positions or a robot may have to respond to a stream of camera images. It is infeasible to learn a complete mapping from every possible input to every possible output. Therefore, a learning agent will typically use some means of grouping input signals. In the case of the BOXES pole balancer, Michie and Chambers discretized the state space. The state is characterized by the cart's position and velocity and the pole's angle and angular velocity. These parameters create a four-dimensional space, which was broken into three regions (left, center, right) for the pole angle, five for the angular velocity, and three for the cart position and velocity. These choices were arbitrary and other combinations also worked.

Having divided the input space into non-overlapping regions, Michie and Chambers associated a push-left and push-right action with each region, or box. The learning algorithm maintains a score for each action and chooses the next action based on that score. BOXES was an early, and simple example, of creating an internal representation for mapping inputs to outputs. The problem with this method is that the structure of the decision-making system is fixed at the start and the learner is incapable of changing the representation. This may be needed if, for example, the subdivisions that were chosen do not correspond to a real decision boundary. A learning system that could adapt its representation has an advantage, in this case.

The BOXES representation can be thought of as a lookup table that implements a function that maps an input to an output. The fixed lookup table can be replaced by a function approximator that, given examples from the desired function, generalizes from them to construct an approximation of that function. Different function approximation techniques can be used. For example, Moore's (1990) function approximator was a ▶ nearest-neighbor algorithm, implemented using kd-tree to improve efficiency. Other function approximation methods may also be used, e.g., Albus' CMAC algorithm (1975), ▶ locally weighted regression (Atkeson et al. 1997), ▶ perceptrons Rosenblatt (1962), multi-layer networks (Hinton et al. 1985), ▶ radial basis functions, etc. Structural credit assignment is also addressed in the creation of hierarchical representations. See ▶ hierarchical reinforcement learning. Other approaches to structural credit assignment include ▶ Value function approximation (Bertsekas and Tsitsiklis 1996) and automatic basis generation (Mahadevan 2009). See the entry on ▶ Gaussian Processes for examples of recent Bayesian and kernel method based approaches to solving the credit assignment problem.

Temporal Credit Assignment

In the pole balancing example described above, the learning system receives a signal when the pole has fallen over. How does it know which actions leading up to the failure contributed to the fall? The system will receive a high-level punishment in the event of a failure or a reward in tasks where there is a goal to be achieved. In either case, it makes sense to assign the greatest credit or blame to the most recent actions and assign progressively less to the preceding actions. Each time a learning trial is repeated, the value of an action is updated so that if it leads to another action of higher value, its weight is increased. Thus, the reward or punishment propagates back through the sequence of decisions taken by the system. The credit assignment problem was addressed by Michie and Chambers, in the BOXES, algorithm but many other solutions have subsequently been proposed. See the entries on ▶ Q-learning (Watkins 1989, 1992) and ▶ temporal difference learning (Barto et al. 1983; Sutton 1984).

Although temporal credit assignment is usually associated with reinforcement learning, it also appears in other forms of learning. In learning by imitation or ▶ behavioral cloning, an agent observes the actions of another agent and tries to learn from traces of behaviors. In this case, the learner must judge which actions of the other agent should receive credit or blame. Plan learning also encounters the same problem (Benson 1995; Wang 1996), as does ▶ explanation-based learning (Mitchell et al. 1986; Dejong and Mooney 1986; Laird et al. 1987).

To illustrate the connection with explanation-based learning, we use one of the earliest examples of this kind of learning, Mitchell and Utgoff's, LEX program (Mitchell et al. 1983). The program was intended to learn heuristics for performing symbolic integration. Given a mathematical expression that included an integral sign, the program tried to transform the expression into one they did not. The standard symbolic integration operators were known to the program but not when it is best to apply them. The task of the learning system was to learn the heuristics for when to apply the operators. This was done by experimentation. If no heuristics were available, the program attempted a brute force search. If the search was successful, all the operators applied, leading to the success were assumed to be positive examples for a heuristic, whereas operators applied during a failed attempt became negative examples. Thus, LEX performed a simple form of credit assignment, which is typical of any system that learns how to improve sequences of decisions.

▶ Genetic algorithms can also be used to evolve rules that perform sequences of actions (Holland 1986). When situation-action rules are applied in a sequence, we have a credit assignment problem that is similar to when we use a reinforcement learning. That is, how do we know which rules were responsible for success or failure and to what extent? Grefenstette (1988) describes a *bucket brigade* algorithm in which rules are given strengths that are adjusted to reflect credit or blame. This is similar to temporal difference learning except that in the bucket brigade the strengths apply to rules rather than states. See Classifier Systems and for a more comprehensive survey of bucket brigade methods, see Goldberg (1989).

Transfer Learning

After a person has learned to perform some task, learning a new, but related, task is usually easier because knowledge of the first learning episode is *transferred* to the new task. *Transfer Learning* is particularly useful for acquiring new concepts or behaviors when given only a small amount for training data. It can be viewed as a form of credit assignment because successes or failures in previous learning episodes bias future learning. Reid (2004, 2007) identifies three forms of ▶ inductive bias involved in transfer learning for rules: language bias, which determines what kinds of rules can be constructed by the learner; the search bias, which determines the order in which rules will be searched; and the evaluation bias, which determines how the quality of the rules will be assessed. Note that learning language bias is a form of structural credit assignment. Similarly, where rules are applied sequentially, evaluation bias becomes temporal credit assignment. Taylor and Stone (2009) give a comprehensive survey of transfer in ▶ reinforcement learning, in which they describe a variety of techniques for transferring the structure of an RL task from one case to another. They also survey methods for transferring evaluation bias.

Transfer learning can be applied in many different settings. Caruana (1997) developed a system for transferring inductive bias in ▶ neural networks performing multitask learning and more recent research has been directed toward transfer learning in ▶ Bayesian Networks (Niculescu and Caruana 2007).

See Transfer Learning and Silver et al. (2005) and Banerjee et al. (2006) for recent work on transfer learning.

Cross-References

▶ Bayesian Network
▶ Classifier Systems
▶ Genetic Programming
▶ Hierarchical Reinforcement Learning
▶ Inductive Bias
▶ Locally Weighted Regression for Control
▶ Nearest Neighbor
▶ Precision
▶ Radial Basis Function Networks
▶ Reinforcement Learning
▶ Temporal Difference Learning

Recommended Reading

Albus JS (1975) A new approach to manipulator control: the cerebellar model articulation controller (CMAC). J Dyn Syst Measur Control Trans ASME 97(3):220–227
Anderson CW, Miller WT (1991) A set of challenging control problems. In: Miller W, Sutton RS, Werbos PJ (eds) Neural networks for control. MIT Press, Cambridge
Atkeson C, Schaal S, Moore A (1997) Locally weighted learning. AI Rev 11:11–73
Banerjee B, Liu Y, Youngblood GM (eds) (2006) Proceedings of the ICML workshop on "structural knowledge transfer for machine learning, Pittsburgh
Barto A, Sutton R, Anderson C (1983) Neuron-like adaptive elements that can solve difficult learning control problems. IEEE Trans Syst Man Cybern SMC-13:834–846
Benson S, Nilsson NJ (1995) Reacting, planning and learning in an autonomous agent. In: Furukawa K,

Michie D, Muggleton S (eds) Machine intelligence, vol 14. Oxford University Press, Oxford
Bertsekas DP, Tsitsiklis J (1996) Neuro-dynamic programming. Athena Scientific, Nashua
Caruana R (1997) Multitask learning. Mach Learn 28:41–75
Dejong G, Mooney R (1986) Explanation-based learning: an alternative view. Mach Learn 1:145–176
Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley Longman Publishing, Boston
Grefenstette JJ (1988) Credit assignment in rule discovery systems based on genetic algorithms. Mach Learn 3(2–3):225–245
Hinton G, Rumelhart D, Williams R (1985) Learning internal representation by back-propagating errors. In: Rumelhart D, McClelland J, Group TPR (eds) Parallel distributed computing: explorations in the microstructure of cognition, vol 1. MIT Press, Cambridge, pp 31–362
Holland J (1986) Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning: an artificial intelligence approach, vol 2. Morgan Kaufmann, Los Altos
Laird JE, Newell A, Rosenbloom PS (1987) SOAR: an architecture for general intelligence. Artif Intell 33(1):1–64
Mahadevan S (2009) Learning representation and control in Markov decision processes: new frontiers. Found Trends Mach Learn 1(4):403–565
Michie D, Chambers R (1968) Boxes: an experiment in adaptive control. In: Dale E, Michie D (eds) Machine intelligence, vol 2. Oliver and Boyd, Edinburgh
Minsky M (1961) Steps towards artificial intelligence. Proc IRE 49:8–30
Mitchell TM, Keller RM, Kedar-Cabelli ST (1986) Explanation based generalisation: a unifying view. Mach Learn 1:47–80
Mitchell TM, Utgoff PE, Banerji RB (1983) Learning by experimentation: acquiring and refining problem-solving heuristics. In: Michalski R, Carbonell J, Mitchell T (eds) Machine kearning: an artificial intelligence approach. Tioga, Palo Alto
Moore AW (1990) Efficient memory-based learning for robot control. Ph.D. thesis, UCAM-CL-TR-209, Computer Laboratory, University of Cambridge, Cambridge
Niculescu-mizil A, Caruana R (2007) Inductive transfer for Bayesian network structure learning. In: Proceedings of the 11th international conference on AI and statistics (AISTATS 2007), San Juan
Reid MD (2004) Improving rule evaluation using multitask learning. In: Proceedings of the 14th international conference on inductive logic programming, Porto, pp 252–269
Reid MD (2007) DEFT guessing: using inductive transfer to improve rule evaluation from limited data. Ph.D. thesis, School of Computer Science and Engineering, The University of New South Wales, Sydney
Rosenblatt F (1962) Principles of neurodynamics: perceptrons and the theory of Brain mechanics. Spartan Books, Washington, DC
Samuel A (1959) Some studies in machine learning using the game of checkers. IBM J Res Develop 3(3):210–229
Silver D, Bakir G, Bennett K, Caruana R, Pontil M, Russell S et al (2005) NIPS workshop on "inductive transfer: 10 years later", Whistler
Sutton R (1984) Temporal credit assignment in reinforcement learning. Ph.D. thesis, Department of Computer and Information Science, University of Massachusetts, Amherst
Sutton R, Barto A (1998) Reinforcement learning: an introduction. MIT Press, Cambridge
Taylor ME, Stone P (2009) Transfer learning for reinforcement learning domains: a survey. J Mach Learn Res 10:1633–1685
Wang X, Simon HA, Lehman JF, Fisher DH (1996) Learning planning operators by observation and practice. In: Proceedings of the second international conference on AI planning systems (AIPS-94), Chicago, pp 335–340
Watkins C (1989) Learning with delayed rewards. Ph.D. thesis, Psychology Department, University of Cambridge, Cambridge
Watkins C, Dayan P (1992) Q-learning. Mach Learn 8(3–4):279–292

Cross-Language Document Categorization

Document Categorization is the task consisting in assigning a document to zero, one or more categories in a predefined taxonomy. *Cross-language document categorization* describes the specific case in which one is interested in automatically categorize a document in a same taxonomy regardless of the fact that the document is written in one of several languages. For more details on the methods used to perform this task see ▸ cross-lingual text mining.

Cross-Language Information Retrieval

Cross-language information retrieval (CLIR) is the task consisting in recovering the subset of a document collection D relevant to a query q, in the special case in which D contains documents written in more than one language. Generally, it is additionally assumed that the subset of relevant documents must be returned as an ordered list, in decreasing order of relevance. For more details on methods and applications see ▸ cross-lingual text mining.

Cross-Language Question Answering

Question answering is the task consisting in finding in a document collection the answer to a question. CLCat is the specific case in which the question and the documents can be in different languages. For more details on the methods used to perform this task see ▸ cross-lingual text mining.

Cross-Lingual Text Mining

Nicola Cancedda and Jean-Michel Renders
Xerox Research Centre Europe, Meylan, France

Definition

Cross-lingual text mining is a general category denoting tasks and methods for accessing the information in sets of documents written in several languages, or whenever the language used to express an information need is different from the language of the documents. A distinguishing feature of cross-lingual text mining is the necessity to overcome some language translation barrier.

Motivation and Background

Advances in mass storage and network connectivity make enormous amounts of information easily accessible to an increasingly large fraction of the world population. Such information is mostly encoded in the form of running text which, in most cases, is written in a language different from the native language of the user. This state of affairs creates many situations in which the main barrier to the fulfillment of an information need is not technological but linguistic. For example, in some cases the user has some knowledge of the language in which the text containing a relevant piece of information is written, but does not have a sufficient control of this language to express his/her information needs. In other cases, documents in many different languages must be categorized in a same categorization schema, but manually categorized examples are available for only one language.

While the automatic translation of text from a natural language into another (machine translation) is one of the oldest problems on which computers have been used, a palette of other tasks has become relevant only more recently, due to the technological advances mentioned above. Most of them were originally motivated by needs of government Intelligence communities, but received a strong impulse from the diffusion of the World-Wide Web and of the Internet in general.

Tasks and Methods

A number of specific tasks fall under the term of Cross-lingual text mining (CLTM), including:

- *Cross-language information retrieval*
- *Cross-language document categorization*
- *Cross-language document clustering*
- *Cross-language question answering*

These tasks can in principle be performed using methods which do not involve any ▸ Text Mining, but as a matter of fact all of them have been successfully approached relying on the statistical analysis of multilingual document collections, especially *parallel corpora*. While CLTM tasks differ in many respect, they are all characterized by the fact that they require to reliably measure the similarity of two text spans written in different languages. There are essentially two families of approaches for doing this:

1. In *translation-based* approaches one of the two text spans is first translated into the language of the other. Similarity is then computed based on any measure used in mono-lingual cases. As a variant, both text spans can be translated in a third *pivot* language.
2. In *latent semantics* approaches, an abstract vector space is defined based on the statistical properties of a *parallel corpus* (or, more rarely, of a *comparable corpus*). Both text spans are then represented as vectors in such *latent semantic* space, where any similarity measure for vector spaces can be used.

The rest of this entry is organized as follows: first Translation-related approaches will be introduced, followed by Latent-semantic approaches. Finally, each of the specific CLTM tasks will be discussed in turn.

Translation-Based Approaches

The simplest approach consists in using a manually-written machine-readable bilingual dictionary: words from the first span are looked up and replaced with words in the second language (see e.g., Zhang and Vines 2005). Since typically dictionaries contain entries for "citation forms" only (e.g., the singular for nouns, the infinitive for verbs etc.), words in both spans are preliminarily *lemmatized*, i.e., replaced with the corresponding citation form. In all cases when the lexica and morphological analyzers required to perform lemmatization are not available, a frequently adopted crude alternative consists in *stemming* (i.e., truncating by taking away a suffix) both the words in the span to be translated and in the corresponding side in the lexicon. Some languages (e.g., Germanic languages) are characterized by a very productive *compounding*: simpler words are connected together to form complex words. Compound words are rarely in dictionaries as such: in order to find them it is first necessary to break compounds into their elements. This can be done based on additional linguistic resources or by means of heuristics, but in all cases it is a challenging operation in itself. If the method used afterward to compare the two spans in the target language can take weights into account, translations are "normalized" in such a way that the cumulative weight of all translations of a word is the same regardless of the number of alternative translations. Most often, the weight is simply distributed uniformly among all alternative translations. Sometimes, only the first translation for each word is kept, or the first two or three.

A second approach consists in extracting a bilingual lexicon from a *parallel corpus* instead of using a manually-written one. Methods for extracting probabilistic lexica look at the frequencies with which a word s in one language was translated with a word t to estimate the translation probability $p(t|s)$. In order to determine which word is the translation of which other word in the available examples, these examples are preliminarily aligned, first at the sentence level (to know what sentence is the translation of what other sentence) and then at the word level. Several methods for aligning sentences at the word level have been proposed, and this problem is a lively research topic in itself (see Brown et al. 1993 for a seminal paper).

Once a probabilistic bilingual dictionary is available, it can be used much in the same way as human-written dictionaries, with the notable difference that the estimated conditional probabilities provide a natural way to distribute weight across translations. When the example documents used for extracting the bilingual dictionaries are of the same style and domain as the text spans to be translated, this can result in a significant increase in accuracy for the final task, whatever this is.

It is often the case that a parallel corpus sufficiently similar in topic and style to the spans to be translated is unavailable, or it is too small to be used for reliably estimating translation probabilities. In such cases, it can be possible to replace or complement the parallel corpus with a "comparable" corpus. A comparable corpus is a pair of collections of documents, one in each of the languages of interest, which are known to be similar in content, although not the translation of one another. A typical case might be two sets of articles from corresponding sections of different newspapers collected during a same period of time. If some additional bilingual *seed* dictionary (human-written or extracted from a parallel corpus) is also available, then the comparable corpus can be leveraged as well: a word t is likely to be the translation of a word s if it turns out that the words often appearing near s are translations of the words often appearing near t. Using this observation it is thus possible to estimate the probability that t is a valid translation of s even though they are not contained in the original dictionary. Most approaches proceed by associating with s a *context vector*. This vector, with one component for each word in the source language, can simply be formed by summing together the count histograms of the words occurring within a fixed window centered in all occurrences of s in the corpus, but is often constructed using statistically more robust association measures, such as mutual information. After a possible normalization step, the context vector $CV(s)$ is translated using the seed dictionary into the target language. A context vector is also extracted from the corpus for all target words t. Eventually, a translation score between s and t is computed as $\langle Tr(CV(s)), CV(t)\rangle$:

$$\begin{aligned}\mathcal{S}(s,t) &= \langle CV(s), Tr(CV(t))\rangle \\ &= \sum_{(s',t')\in\mathcal{D}} a(s,s')a(t,t'),\end{aligned}$$

where a is the association score used to construct the context vector. While effective in many cases, this approach can provide inaccurate similarity values when polysemous words and synonyms appear in the corpus. To deal with this problem, Gaussier et al. (2004) propose the following extension:

$$\mathcal{S}(s,t) = \sum_{(s',t')\in\mathcal{D}} \left(\sum_{s'} a(s's'')a(s,s'')\right) \left(\sum_{t''} a(t',t'')a(t,t'')\right),$$

which is more robust in cases when the entries in the seed bilingual dictionary do not cover all senses actually present in the two sides of the comparable corpus.

Although these methods for building bilingual dictionaries can be (and often are) used in isolation, it can be more effective to combine them.

Using a bilingual dictionary directly is not the only way for translating a span from one language into another. A second alternative consists in using a *machine translation* (MT) system. While the MT system, in turn, relies on a bilingual dictionary of some sort, it is in general in the position of leveraging contextual clues to select the correct words and put them in the right order in the translation. This can be more or less useful depending on the specific task. MT systems fall, broadly speaking, into two classes: rule-based and statistical. Systems in the first class rely on sets of hand-written rules describing how words and syntactic structures should be translated. Statistical machine translation (SMT) systems learn this mapping by performing a statistical analysis of a parallel corpus. Some authors (e.g., Savoy and Berger 2005) also experimented with combining translation from multiple machine translation systems.

Latent Semantic Approaches

In CLTM, *Latent Semantic* approaches rely on some interlingua (language-independent) representation. Most of the time, this interlingua representation is obtained by linear or non-linear statistical analysis techniques and more specifically ▸ dimensionality reduction methods with ad-hoc optimization criterion and constraints. But, others adopt a more manual approach by exploiting multilingual thesauri or even multilingual ontologies in order to map textual objects towards a list – possibly weighted – of interlingua concepts.

For any textual object (typically a document or a section of document), the *interlingua* concept representation is derived from a sequence of operations that encompass:

1. Linguistic preprocessing (as explained in previous sections, this step amounts to extract the relevant, normalized "terms" of the textual objects, by tokenisation, word segmentation/decompounding, lemmatisation/stemming, part-of-speech tagging, stopword removal, corpus-based term filtering, Noun-phrase extractions, etc.).
2. Semantic enrichment and/or monolingual dimensionality reduction.
3. Interlingua semantic projection.

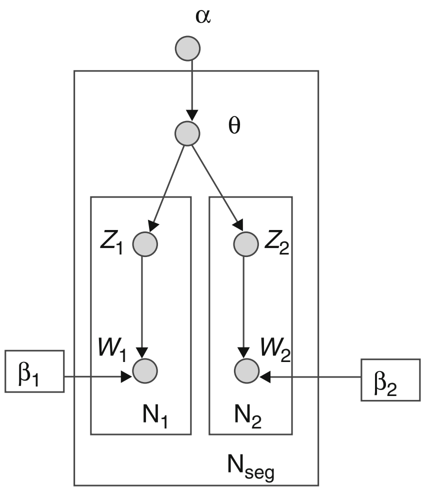
A typical semantic enrichment method is the *generalized vector space model*, that adds related terms – or neighbour terms – to each term of the textual object, neighbour terms being defined by some co-occurrence measures (for instance, mutual information). Semantic enrichment can alternatively be achieved by using (monolingual) thesaurus, exploiting relationships such as synonymy, hyperonymy and hyponymy. Monolingual dimensionality reduction consists typically in performing some *latent semantic analysis* (LSA), some form of principal component analysis on the textual object/term matrix. Dimensionality reduction techniques such as LSA or their discrete/probabilistic variants such as *probabilistic semantic analysis* (PLSA) and *latent dirichlet allocation* (LDA) offer to some extent a semantic robustness to deal with the effects of polysemy/synonymy, adopting a language-dependent concept representation in a space of dimension much smaller than the size of the vocabulary in a language.

Of course, steps (1) and (2) are highly language-dependent. Textual objects written in different languages will not follow the same linguistic processing or semantic enrichment/dimensionality reduction. The last step (3), however, aims at projecting textual objects in the same language-independent concept space, for any source language. This is done by first extracting these common concepts, typically from a parallel corpus that offers a natural multiple-view representation of the same objects. Starting from these multiple-view observations, common factors are extracted through the use of canonical correlation analysis (CCA), cross-language latent semantic analysis, their kernelized variants (eg. Kernel-CCA) or their discrete, probabilistic extensions (cross-language latent dirichlet allocation, multinomial CCA, . . .). All these methods try to discover latent factors that simultaneously explain as much as possible the "intra-language" variance and the "inter-language" correlation. They differ in the choice of the underlying distributions and how they precisely define and combine these two criteria. The following subsections will describe them in more details.

As already emphasized, CLTM mainly relies on defining appropriate similarities between textual objects expressed in different languages. Numerous categorization, clustering and retrieval algorithms focus on defining efficient and powerful measures of similarity between objects, as strengthened recently by the development of kernel methods for textual information access. We will see that the (linear) statistical algorithms used for performing steps (2) and (3) can most of the time be embedded into one valid (Mercer) kernel, so that we can very easily obtain non-linear variants of these algorithms, just by adopting some standard non-linear kernels.

Cross-Language Semantic Analysis

This amounts to concatenate the vectorial representation of each view of the objects of the parallel collection (typically, objects are aligned sentences), and then to perform standard singular value decomposition of the global object/term matrix. Equivalently, defining the kernel similarity matrix between all pairs of multi-view objects as the sum of the mono-lingual textual similarity matrices, this amounts to perform the eigenvalue decomposition of the corresponding kernel Gram matrix, if a dual formulation is adopted. The number of eigenvalues/eigenvectors that are retained to define the latent factors and the corresponding projections is typically from several hundreds of components to several thousands, still much fewer than the original sizes of the vocabulary. Note that this process does not really control the formation of *interlingua* concepts: nothing



Cross-Lingual Text Mining, Fig. 1 Latent dirichlet allocation of a parallel corpus

prevents the method from extracting factors that are linear combination of terms in one language only.

Cross-Language Latent Dirichlet Allocation

The extraction of *interlingua components* is realised by using LDA to model the set of parallel objects, by imposing the same proportion of components (topics) for all views of the same object. This is represented in Fig. 1.

LDA is performing some form of clustering, with a predefined number of components (K) and with the constraint that the two views of the same object belongs to the clusters with the same membership values. This results in $2.K$ component profiles that are then used for "folding in" (projecting) new documents by launching some form of EM to derive their posterior probabilities to belong to each of the language-independent component. The similarity between two documents written in different languages is obtained by comparing their posterior distribution over these latent classes. Note that this approach could easily integrate supervised topic information and provides a nice framework for semi-supervised *interlingua* concept extraction.

Cross-Language Canonical Correlation Analysis

The Primal Formulation

CCA is a standard statistical method to perform multi-block multivariate analysis, the goal being to find linear combinations of variables for each block (i.e., each language) that are maximally correlated. In other words, CCA is able to enforce the commonality of latent concept formations by extracting maximally correlated projections. Starting from a set of paired views of the same objects (typically, aligned sentences of a parallel corpus) in languages L1 and L2, the algebraic formulation of this optimization problem leads to a generalized eigenvalue problem of size (n_1+n_2), where n_1 and n_2 are the sizes of the vocabularies in L1 and L2 respectively. For obvious scalability reasons, the dual – or kernel – formulation (of size N, the number of paired objects in the training set) is often preferred.

Kernel Canonical Correlation Analysis

Basically, Kernel Canonical Correlation Analysis amounts to do CCA on some implicit, but more complex feature space and to express the projection coefficients as linear combination of the training paired objects. This results in the dual formulation, which is a generalized eigenvalue/vector problem of size $2N$, that involves only the monolingual kernel gram matrices K_1 and K_2 (matrices of monolingual textual similarities between all pairs of objects in the training set in language L1 and L2 respectively). Note that it is easy to show that the eigenvalues go by pairs: we always have two symmetrical eigenvalues $+\lambda$ and $-\lambda$. This kernel formulation has the advantage to include any text specific prior properties in the kernel (e.g., use of N-gram kernels, word-sequence kernels, and any semantically-smoothed kernel). After extraction of the first k generalized eigenvalues/eigenvectors, the similarity between any pair of test objects in languages L1 and L2 can be computed by using projection matrices composed of extracted eigenvector as well as the (monolingual) kernels of the test objects with the training objects.

Regularization and Partial Least Squares Solution

When the number of training examples (N) is less than n_1 and n_2 (the dimensions of the monolingual feature spaces), the eigenvalue spectrum of the KCCA problem has generally two null eigenvalues (due to data centering), $(N - 1)$ eigenvalues in $+1$ and $(N - 1)$ eigenvalues in -1, so that, as such, the KCCA problem only results in trivial solutions and is useless. When using kernel methods, the case $(N < n_1, n_2)$ is frequent, so that some regularization scheme is needed. One way of realizing this regularization is to resort to finding the directions of maximum covariance (instead of correlation): this can be considered as a partial least squares (PLS) problem, whose formulation is very similar to the CCA problem. Adopting a mixed criterion CCA/PLS (trying to maximize a combination of covariance and correlation between projections) turns out to both avoid over-fitting (or spurious solutions) and to enhance numerical stability.

Approximate Solutions

Both CCA and KCCA suffer from a lack of scalability, due to the fact the complexity of generalized eigenvalue/vector decomposition is $O(N^3)$ for KCCA or $O(\min(n_1, n_2)^3)$ for CCA. As it can be shown that performing a complete KCCA (or KPLS) analysis amounts to do first complete PCA's, and then a linear CCA (or PLS) on the resulting new projections, it is obvious that we could reduce the complexity by working on a reduced-rank approximation (incomplete KPCA) of the kernel matrices. However, the implicit projections derived from incomplete KPCA may be not optimal with respect to cross-correlation or covariance criteria. Another idea to decrease the complexity is to perform some incomplete Cholesky decomposition of the (monolingual) kernel matrices K_1 and K_2 (that is equivalent to partial Gram-Schmit orthogonalisation in the feature space): $K_1 = G_1.\, G_1^t$ and $K_2 = G_2.\, G_2^t$, with G_i of rank $k \ll N$. Considering G_i as the new representation of the training data, KCCA now reduces to solving a generalized eigenvalue problem of size $2.k$.

Specific Applications

The previous sections illustrated a number of different ways of solving the core problem of cross-language text mining: quantifying the similarity between two spans of text in different languages. In this section we turn to describing some actual applications relying on these methods.

Cross-Language Information Retrieval (CLIR)

Given a collection of documents in several languages and a single query, the CLIR problem consists in producing a single ranking of all documents according to their relevance to the query. CLIR is in particular useful whenever a user has some knowledge of the languages in which documents are written, but not enough to express his/her information needs in those languages by means of a precise query. Sometimes CLIR engines are coupled with translation tools to help the user access the content of relevant documents written in languages unknown to him/her. In this case document collections in an even larger number of languages can be effectively queried.

It is probably fair to say that the vast majority of the CLIR systems use a translation-based approach. In most cases it is the query which is translated in all languages before being sent to monolingual search engines. While this limits the amount of translation work that needs be done, it requires doing it on-line at query time. Moreover, when queries are short it can be difficult to translate them correctly, since there is little context to help identifying the correct sense in which words are used. For these reasons several groups also proposed translating all documents at indexing time instead. Regardless of whether queries or documents are translated, whenever similarity scores between (possibly translated) queries and (possibly translated) documents are not directly comparable, all methods then face the problem of merging multiple monolingual rankings in a single multilingual ranking.

Research in CLIR and cross-language question answering (see below) has been significantly stimulated by at least three government-sponsored evaluation campaigns:

- The NII Test Collection for IR Systems (NTCIR) (http://research.nii.ac.jp/ntcir/), running yearly since 1999, focusing on Asian languages (Japanese, Chinese, Korean) and English.
- The Cross-Language Evaluation Forum (CLEF) (http://www.clef-campaign.org), running yearly since 2000, focusing on European languages.
- A cross-language track at the Text Retrieval Conference (TREC) (http://trec.nist.gov/), which was run until 2002, focused on querying documents in Arabic using queries in English.

The respective websites are ideal starting points for any further exploration on the subject.

Cross-Language Question Answering (CLQA)

Question answering is the task of automatically finding the answer to a specific question in a document collection. While in practice this vague description can be instantiated in many different ways, the sense in which the term is mostly understood is strongly influenced by the task specification formulated by the National Institute of Science and Technology (NIST) of the United States for its TREC evaluation conferences (see above). In this sense, the task consists in identifying a *text snippet*, i.e., a substring, of a predefined maximal length (e.g., 50 characters, or 200 characters) within a document in the collection containing the answer. Different classes of questions are considered:

- Questions around facts and events.
- Questions requiring the definition of people, things and organizations.
- Questions requiring as answer lists of people, objects or data.

Most proposals for solving the QA problem proceed by first identifying promising documents (or document segments) by using information retrieval techniques treating the question as a query, and then performing some finer-grained analysis to converge to a sufficiently short snippet. Questions are classified in a hierarchy of possible "question types." Also, documents are preliminarily indexed to identify elements (e.g., person names) that are potential answers to questions of relevant types (e.g., "Who" questions).

Cross-language question answering (CLQA) is the extension of this task to the case where the collection contains documents in a language different than the language of the question. In this task a CLIR step replaces the monolingual IR step to shortlist promising documents. The classification of the question is generally done in the source language.

Both CLEF and NTCIR (see above) organize cross-language question answering comparative evaluations on an annual basis.

Cross-Language Categorization (CLCat) and Clustering (CLCLu)

Cross-language categorization tackles the problem of categorizing documents in different languages in a same categorization scheme.

The vast majority of document categorization systems rely on machine learning techniques to automatically acquire the necessary knowledge (often referred to as a *model*) from a possibly large collection of manually categorized documents. Most often the model is based on frequency counts of words, and is thus intrinsically language-dependent. The most direct way to perform categorization in different languages would consist in manually categorizing a sufficient amount of documents in all languages of interest and then train a set of independent categorizer. In some cases, however, it is impractical to manually categorize a sufficient number of documents to ensure accurate categorization in all languages, while it can be easier to identify bilingual dictionaries or parallel (or comparable) corpora for the language pairs and in the application domain of interest. In such cases it is then preferable to obtain manually categorized documents only for a single language A and use them to train a monolingual categorizer. Any of the translation-based approaches described above can then be used to translate a document originally in language B – or most often its representation as a *bag of words*– into language A. Once the document is translated, it can be categorized using the monolingual A system.

As an alternative, latent-semantics approaches can be used as well. An existing parallel corpus can be used to identify an abstract vector space common to A and B. The manually categorized documents in A can then be represented in this space, and a model can be learned which operates directly on this latent-semantic representation. Whenever a document in B needs be categorized, it is first projected in the common semantic space and then categorized using the same model.

All these considerations carry unchanged to the cross-language clustering task, which consists in identifying subsets of documents in a multilingual document collection which are mutually similar to one another according to some criterion. Again, this task can be effectively solved by either translating all documents into a single language or by learning a common semantic space and performing the clustering task there.

While CLCat and Clustering are relevant tasks in many real-world situations, it is probably fair to say that less effort has been devoted to them by the research community than to CLIR and CLQA.

Recommended Reading

Brown PE, Della Pietra VJ, Della Pietra SA, Mercer RL (1993) The mathematics of statistical machine translation: parameter estimation. Comput Linguist 12(2):263–311

Gaussier E, Renders J-M, Matveeva I, Goutte C, Déjean H (2004) A geometric view on bilingual lexicon extraction from comparable corpora. In: Proceedings of the 42nd annual meeting of the association for computational linguistics, Barcelona. Association for Computational Linguistics, Morristown

Savoy J, Berger PY (2005) Report on CLEF-2005 evaluation campaign: monolingual, bilingual and GIRT information retrieval. In: Proceedings of the cross-language evaluation forum (CLEF). Springer, Heidelberg, pp 131–140

Zhang Y, Vines P (2005) Using the web for translation disambiguation. In: Proceedings of the NTCIR-5 workshop meeting, Tokyo

Cross-Validation

Definition

Cross-validation is a process for creating a distribution of pairs of training and ▸ test sets out of a single ▸ data set. In cross validation the data are partitioned into k subsets, $S_1 \ldots S_k$, each called a *fold*. The folds are usually of approximately the same size. The learning algorithm is then applied k times, for $i = 1$ to k, each time using the union of all subsets other than S_i as the ▸ training set and using S_i as the ▸ test set.

Cross-References

▸ Algorithm Evaluation
▸ Leave-One-Out Cross-Validation

Cumulative Learning

Pietro Michelucci[1] and Daniel Oblinger[2]
[1]Strategic Analysis, Inc., Arlington, VA, USA
[2]DARPA/IPTO, Arlington, VA, USA

Synonyms

Continual learning; Lifelong learning; Sequential inductive transfer

Definition

Cumulative learning (CL) exploits knowledge acquired on prior tasks to improve learning performance on subsequent related tasks. Consider, for example, a CL system that is learning to play chess. Here, one might expect the system to learn from prior games concepts (e.g., favorable board positions, standard openings, end games, etc.) that can be used for future learning. This is in contrast to base learning (Vilalta and Drissi 2002) in which a fixed learning algorithm is applied to a single task and performance tends to improve only with more exemplars. So, in CL there tends to be explicit reuse of learned knowledge to constrain new learning, whereas base learning depends entirely upon new external inputs.

Relevant techniques for CL operate over multiple tasks, often at higher levels of abstraction, such as new problem space representations, task-based selection of learning algorithms, dynamic adjustment of learning parameters, and iterative analysis and modification of the learning algorithms themselves. Though actual usage of this term is varied and evolving, CL typically connotes sequential ▶ inductive transfer. It should be noted that the word "inductive" in this connotation qualifies the transfer of knowledge to new tasks, not the underlying learning algorithms.

Related Terminology

The terms "meta-learning" and "learning to learn" are sometimes used interchangeably with CL. However each of these concepts has a specific relationship to CL.

▶ Meta-learning (Brazdil et al. 2009; Vilalta and Drissi 2002) involves the application of learning algorithms to meta-data, which are abstracted representations of input data or learning system knowledge. In the case that abstractions of system knowledge are themselves learning algorithms, meta-learning involves assessing the suitability of these algorithms for previous tasks and, on that basis, selecting algorithms for new tasks (see entry on "▶ Metalearning"). In general, the sharing of abstracted knowledge across tasks in a CL system implies the use of meta-learning techniques. However, the converse is not true. Meta-learning can and does occur in learning systems that do not accumulate and transfer knowledge across tasks.

Learning to learn is a synonym for inductive transfer. Thus, learning to learn is more general than CL. Though it specifies the application of knowledge learned in one domain to another, it does not stipulate whether that knowledge is accumulated and applied sequentially or shared in a parallel learning context.

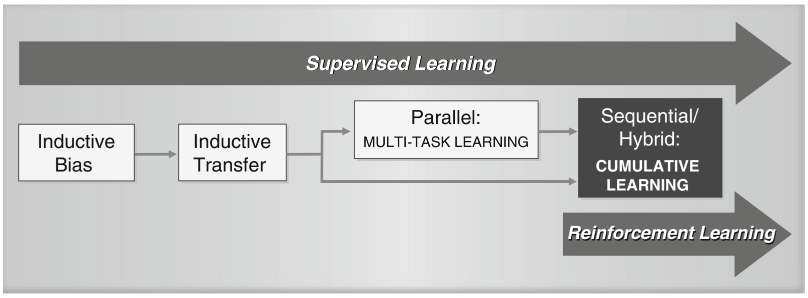
Motivation and Background

Traditional ▶ supervised learning approaches require large datasets and extensive training in order to generalize to new inputs in a single task. Furthermore, traditional (non-CL) ▶ reinforcement learning approaches require tightly constrained environments to ensure a tractable state space. In contrast, humans are able to generalize across tasks in dynamic environments from brief exposure to small datasets. The human advantage seems to derive from the ability to draw upon prior task and context knowledge to constrain hypothesis development for new tasks. Recognition of this disparity between human learning and traditional machine learning had led to the pursuit of methods that seek to emulate the accumulation and exploitation of task-based knowledge that is observed in humans. A coarse evolution of this work is depicted in Fig. 1.

History

Advancements in CL have resulted from two classes of innovation: the development of techniques for ▶ inductive transfer and the integration of those techniques into autonomous learning systems.

Alan Turing (1950) was the first to propose a cumulative learning system. His 1950 paper is best remembered for the imitation game, later known as the Turing test. However, the final sections of the paper address the question of how a machine could be made sufficiently complex to be able to pass the test. He posited that program-



Cumulative Learning, Fig. 1 Evolution of cumulative learning

ming it would be too difficult a task. Therefore, it should be instructed as one might teach a child, starting with simple concepts and working up to more complex ones.

Banerji (1964) introduced the use of predicate logic as a description language for machine learning. Thus, Banerji was one of the earliest advocates of what would later become ▶ ILP. His concept description language allowed the use of background knowledge and therefore was an extensible language. The first implementation of a cumulative learning system based on Banerjis ideas was Cohens CONFUCIUS (Cohen 1978; Cohen and Sammut 1982). In this work, an instructor teaches the system concepts that are stored in a long-term memory. When examples of a new concept are seen, their descriptions are matched against stored concepts, which allow the system to re-describe the examples in terms of the background knowledge. Thus, as more concepts are accumulated, the system is capable of describing complex objects more compactly than if it had not had the background knowledge. Compact representations generally allow complex concepts to be learned more efficiently. In many cases, learning would be intractable without the prior knowledge. See the entries on ▶ Inductive Logic Programming, which describe the use of background knowledge further.

Independent of the research in symbolic learning, much of the ▶ inductive transfer research that underlies CL took root in ▶ artificial neural network research, a traditional approach to ▶ supervised learning. For example, Abu-Mostafa (1990) introduced the notion of reducing the hypothesis space of a neural network by introducing "hints" either as hard-wired additions to the network or via examples designed to teach a particular invariance. The task of a neural network can be thought of as the determination of a function that maps exemplars into a classification space. So, in this context, hints constitute an articulation of some aspect of the target mapping function. For example, if a neural network is tasked with mapping numbers into primes and composites, one "hint" would be that all even numbers (besides 2) are composite. Leveraging such a priori knowledge about the mapping function may facilitate convergence on a solution. An inherent limitation to neural networks, however, is their immutable architecture, which does not lend itself to the continual accumulation of knowledge. Consequently, Ring (1991) introduced a neural network that constructs new nodes on demand in a reinforcement learning context in order to support ongoing hierarchical knowledge acquisition and transfer. In this model, nodes called "bions" correspond simultaneously to the enactment and perception of a single behavior. If two bions are activated in sequence repeatedly, a new bion is created to join the coincident pair and represent their collective functionality.

Contemporaneously, Pratt et al. (1991) investigated the hypothesis that knowledge acquired by one neural network could be used to assist another neural network learn a related task. In the speech recognition domain, they trained three separate networks, each corresponding to speech segments of a different length, such that each network was optimized to learn certain types of phonemes. They then demonstrated that a direct transfer of information encoded as network weights from these three specialized networks to a single, combined speech recognition network resulted in a tenfold reduction in training epochs for the combined network compared with the number of training epochs required when no knowledge was transferred. This was one of the first empirical results in neural network-based transfer learning. Caruana (1993) extended this work to demonstrate the performance benefits associated with the simultaneous transfer of ▸ inductive bias in a "Multitask Learning" (MTL) methodology. In this work, Caruana hypothesized that training the same neural network simultaneously on related tasks would naturally induce additional constraints on learning for each individual task. The intuition was that converging on a mapping in support of multiple tasks with shared representations might best reveal aspects of the input that are invariant across tasks, thus obviating within-task regularities, which might be less relevant to classification. Those empirical results are supported by Baxter (1995) who proved that the number of examples required by a representation learner for learning a single task is an inverse linear function of the number of simultaneous tasks being learned.

Though the innovative underpinnings of inductive transfer that critically underlie CL evolved in a supervised learning context, it was the integration of those methods with classical reinforcement learning that has led to current models of CL. Early integration of this type comes from Thrun and Mitchell (1995), who applied an extension of explanation-based learning (EBL), called explanation-based neural networks (EBNN) (Mitchell and Thrun 1993), to an agent-based "lifelong learning framework." This framework provides for the acquisition of different control policies for different environments and reward functions. Since the robot actuators, sensors, and the environment (largely) remain invariant, this framework supports the use of knowledge acquired from one control problem to be applied to another. By using EBNN to allow learning from previous control problems to constrain learning on new control problems, learning is accelerated over the lifetime of the robot.

More recently, Silver and Mercer (2002) introduced a hybrid model that involves a combination of parallel and sequential inductive transfer in an autonomous agent framework. The so-called task rehearsal method (TRM) uses MTL to combine new training inputs with relevant exemplars that are generated from prior task knowledge. Thus, inductive bias is achieved by training the neural networks on new tasks while simultaneously rehearsing learned task knowledge.

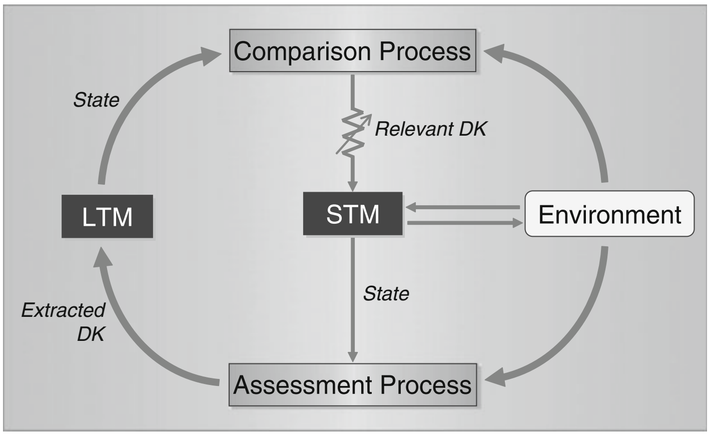
Structure of the Learning System

CL is characterized by systems that use prior knowledge to bias future learning. The canonical interpretation is that knowledge transfer occurs at the task level. Although this description encompasses a broad research space, it is not boundless. In particular, CL systems must be able to (1) retain knowledge and (2) use that knowledge to restrict the hypothesis space for new learning. Nonetheless, learning systems can vary widely across numerous orthogonal dimensions and still meet these criteria.

Toward a CL Specification

Recognizing the empirical utility of a more specific delineation of CL systems, Silver and Poirier (2005) introduced a set of functional requirements, classification criteria, and performance specifications that characterize more precisely the scope of machines capable of lifelong learning. Any system that meets these requirements is considered a machine lifelong learning (ML3) system. A general CL architecture that conforms to the ML3 standard is depicted in Fig. 2.

Two basic memory constructs are typical of CL systems. Long term memory (LTM) is required for storing domain knowledge (DK) that can be used to bias new learning. Short term memory (STM) provides a working memory for building representations and testing hypotheses associated with new task learning. Most of the ML3 requirements specify the interplay of these constructs.



Cumulative Learning, Fig. 2 Typical CL system

LTM and STM are depicted in Fig. 2, along with a comparison process, an assessment process, and the learning environment. In this model, the comparison process evaluates the training input in the context of LTM to determine the most relevant domain knowledge that can be used to constrain short term learning. The comparison process also determines the weight assigned to domain knowledge that is used to bias short term learning. Once the rate of performance improvement on the primary task falls below a threshold the assessment process compares the state of STM to the environment to determine which domain knowledge to extract and store in LTM.

Classification of CL Systems

The simplicity of the architecture shown in Fig. 2 belies the richness of the feature space for CL systems. The following classification dimensions are derived largely from the ML3 specification. This list includes both qualitative and quantitative dimensions. They are presented in three overlapping categories: architectural features, characteristics of the knowledge base, and learning capabilities.

Architecture

The following architectural dimensions for a CL system range from paradigm choices to low-level interface considerations.

Learning paradigm – The learning paradigm(s) may include supervised learning (e.g., neural network, SVM, ILP, etc.), unsupervised learning (e.g., clustering), reinforcement learning (e.g., automated agent), or some combination thereof. Figure 2 depicts a general architecture with processes that are common across these learning paradigms, and which could be elaborated to reflect the details of each.

Task order – CL systems may learn tasks sequentially (Thrun and Mitchell 1995), in parallel (e.g., Caruana 1993), or via a hybrid methodology (e.g., TRM Silver and Mercer 2002). One hybrid approach is to engage in practice (i.e., revisiting prior learned tasks). Transferring knowledge between learned tasks through practice may serve to improve generalization accuracy. Task order would be reflected in the sequence of events within and among process arrows in the Fig. 2 architecture. For example, a system may alternate between processing new exemplars and "practicing" with old, stored exemplars.

Transfer method – Knowledge transfer can also be representational or functional. Functional transfer provides implicit pressure from related training exemplars. For example, the environmental input in Fig. 2 may take the form of training exemplars drawn randomly from data representing two related tasks, such that learning to classify exemplars from one task implicitly constrains learning on the other task. Representational knowledge transfer involves the direct or indirect (Pratt et al. 1991) assignment of a hypothesis representation. A direct inductive transfer entails the assignment of an original hypothesis representation, such as a vector of trained neural network activation weights. This might take the form of a direct injection to LTM in Fig. 2. Indirect transfer implies that some level of abstraction analysis has been applied to the hypothesis representation prior to assignment.

Learning stages – A learning system may implement learning in a single stage or in a series of stages. An example of a two-stage system is one that waits to initiate the long-term storage of domain knowledge until after primary task learning in short-term memory is complete. Like task order, learning stages would be reflected in the sequence of events within and among process arrows in the Fig. 2 architecture. But in this case, ordering pertains to the manner in which learning is staged across encoding processes.

Interface cardinality – The interface cardinality can be fixed or variable. Fixing the number of inputs and outputs has the advantage of providing a consistent interface without posing restrictions on the growth of the internal representation.

Data type – The input and output data types can be fixed or variable. A type-flexible system can produce both categorical and scalar predictions.

Scalability – CL systems may or may not scale on a variety of dimensions including inputs, outputs, training examples, and tasks.

Knowledge

This category pertains to the long-term storage of learned knowledge. Thus, the following CL dimensions characterize knowledge representation, storage, and retrieval.

Knowledge representation – Stored knowledge can manifest as functional or representational. Functional knowledge retention involves the storage of specific exemplars or parameter values, which tends to be more accurate, whereas representational knowledge retention involves the storage of hypotheses derived from training on exemplars, which has the advantage of storage economy.

Retention efficacy – The efficacy of long term retention varies across CL systems. Effective retention implies that only domain knowledge with an acceptable level of accuracy is retained so that errors aren't propagated to future hypotheses. A related consideration is whether or not the consolidation of new domain knowledge degrades the accuracy of current or prior hypotheses.

Retention efficiency – The retention efficiency of long term memory can vary according to both economy of representation and computationally efficiency.

Indexing method – The input to the comparison process used to select appropriate knowledge for biasing new learning may simply be exemplars (as provided by LTM in Fig. 2) or may take a representational form (e.g., a vector of neural network weights).

Indexing efficiency – CL systems vary in terms of the speed and accuracy with which they can identify related prior knowledge that is suitable for inductive transfer during short term learning. The input to this selection process is the indexing method.

Meta-knowledge – CL systems differentially exhibit the ability to abstract, store, and utilize meta-knowledge, such as characteristics of the input space, learning system parameter values, etc.

Learning

While all of the dimensions listed herein impact learning, the following dimensions correspond to specific learning capabilities or learning performance metrics.

Agency – The agency of a learning system is the degree of sophistication exhibited by its top-level controller. For example a learning system may be on the low end of the agency continuum if it always applies one predetermined learning method to one task or on the high end if it selects among many learning methods as a function of the learning task. One might imagine, for example, two process diagrams such as the one depicted in Fig. 2, that share the same LTM, but are otherwise distinct and differentially activated by a governing controller as a function of qualitative aspects of the input.

Utility – Domain knowledge acquisition can be deliberative in the sense that the learning system decides which hypotheses to incorporate based upon their estimated utility, or reflexive, in which case all hypotheses are stored irrespective of utility considerations.

Task awareness – Task awareness characterizes the system's ability to identify the beginning and end of a new task.

Bias modulation – A CL system may have the ability to determine the extent to which short-term learning would benefit from inductive transfer and, on that basis, assign a relevant weight. The depth of this analysis can vary and might consider factors such as the estimated sample complexity, number of exemplars, the generalization accuracy of retained knowledge, and relatedness of retained knowledge.

Learning efficacy – A measure of learning efficacy is derived by comparing generalization performance in the presence and absence of an inductive bias. Learning is considered effective when the application of an inductive bias results in greater generalization performance on the primary task than when the bias is absent.

Learning efficiency – Similarly, learning efficiency is assessed by comparing the computational time needed to generate a hypothesis in the presence and absence of an inductive bias. Lower computational time in the presence of bias signifies greater learning efficiency.

The Research Space

Table 1 summarizes the classification dimensions, providing an overview of the research space, an evaluative framework for assessing and contrasting CL approaches, and a generative framework for identifying new areas of exploration. In addition, checked items in the Values column indicate ML3 guidance. Specifically, an ideal ML3 system would correspond functionally to the called-out items and performance criteria. However, Silver and Poirier (2005) allude to the fact that it would be nigh impossible to generate a strictly compliant ML3 system since some of the recommended criteria do not coexist easily. For example, effective and efficient learning are mutually incompatible because they require different forms of knowledge transfer. Nonetheless, a CL system that falls within scope of the majority of the ML3 criteria would be well-positioned to exhibit lifelong learning behavior.

Future Directions

Emergent work (Oblinger 2006; Swarup et al. 2006) in instructable computing has given rise to a new CL paradigm that is largely ML3 compliant and involves high degrees of task awareness and agency sophistication. Swarup et al. (2006) describe an approach in which domain knowledge is represented in the form of structured graphs. Short term (primary task) learning occurs via a genetic algorithm, after which domain knowledge is extracted by mining frequent subgraphs. The accumulated domain knowledge forms an ontology to which the learning system grounds symbols as a result of structured interactions with instructional agents. Subsequent interactions occur using the symbol system as a shared lexicon for communication between the instructor and the learning system. Knowledge acquired from these interactions bootstrap future learning.

The Bootstrapped Learning framework proposed by Oblinger (2006) provides for hierarchical, domain-independent learning that, like the effort described above, is also premised on a model of building concepts from structured lessons. In this case, however, there is no a priori knowledge acquisition. Instead, some "common" knowledge about the world is provided explicitly to the learning system, and then lessons are taught by a human teacher using the same natural instruction methods that would be used to teach another human. Rather than requiring a specific learning algorithm, this framework provides a context for evaluating and comparing learning algorithms. It includes a knowledge representation language that supports syntactic, logical, procedural, and functional knowledge, an interaction language for communication among the learning system, instructor, and environment, and an integration architecture that evaluates, processes, and responds to interaction language communiqués in the context of existing knowledge and through the selective utilization of available learning algorithms.

Cumulative Learning, Table 1 CL system dimensions

Category	Dimension	Values (ML3 guidance is indicated by {✓})
Architecture	Learning paradigm	Supervised learning
		Reinforcement learning
		Unsupervised learning
		{✓} Hybrid
	Task order	Sequential
		Parallel
		{✓} Revisit (practice)
		Hybrid
	Transfer method	Functional
		Representational – direct
		Representational – indirect
	Learning stages	{✓} Single (computational retention efficiency)
		Multiple
	Interface cardinality	{✓} Fixed
		Variable
	Data type	Fixed
		Variable
	Scalability	{✓} Inputs
		{✓} Outputs
		{✓} Exemplars
		{✓} Tasks
Knowledge	Representation	Functional
		Representational – disjoint
		{✓} Representational – continuous
	Retention efficacy	{✓} Improves prior task performance
		{✓} Improves new task performance
	Retention efficiency	{✓} Space (memory usage)
		{✓} Time (computational processing)
	Indexing method	{✓} Deliberative – functional
		{✓} Deliberative – representational
		Reflexive
	Indexing efficiency	{✓} Time $< \mathrm{O}(n^c), c > 1 (n = \text{tasks})$
	Meta-knowledge	{✓} Probability distribution of input space
		Learning curve
		Error rate
Learning	Agency	Single learning method
		Task-based selection of learning method
	Utility	Single learning method
		Task-based selection of learning method
	Task awareness	Task boundary identification (begin/end)
	Bias modulation	{✓} Estimated sample complexity
		{✓} Number of task exemplars
		{✓} Generalization accuracy of retained knowledge
		{✓} Relatedness of retained knowledge
	Learning efficacy	{✓} Generalization $\mid$ bias $\geq$ generalization $\mid$ no bias
	Learning efficiency	{✓} Time $\mid$ bias $\leq$ time $\mid$ no bias

The learning performance advantages anticipated by these proposals for instructional computing seem to stem from the economy of representation afforded by hierarchical knowledge combined with the tremendous learning bias imposed by explicit instruction.

Recommended Reading

Abu-Mostafa Y (1990) Learning from hints in neural networks (invited). J Complex 6(2):192–198

Banerji RB (1964) A language for the description of concepts. General Syst 9:135–141

Baxter J (1995) Learning internal representations. In: (COLT): proceeding of the workshop on computational learning theory, Santa Cruz. Morgan Kaufmann

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

Caruana R (1993) Multitask learning: a knowledge-based source of inductive bias. In: Proceedings of the tenth international conference on machine learning, University of Massachusetts, Amherst, pp 41–48

Caruana R (1996) Algorithms and applications for multitask learning. In: Machine learning: proceedings of the 13th international conference on machine learning (ICML 1996), Bari. Morgan Kauffmann, pp 87–95

Cohen BL (1978) A theory of structural concept formation and pattern recognition. Ph.D. thesis, Department of Computer Science, The University of New South Wales

Cohen BL, Sammut CA (1982) Object recognition and concept learning with CONFUCIUS. Pattern Recogn J 15(4):309–316

Mitchell T (1980) The need for biases in learning generalizations. Rutgers TR CBM-TR-117

Mitchell TM, Thrun SB (1993) Explanation-based neural network learning for robot control. In: Hanson CG (eds) Advances in neural information processing systems, vol 5. Morgan-Kaufmann, San Francisco, pp 287–294

Nilsson NJ (1996) Introduction to machine learning: an early draft of a proposed textbook, p 12. Online at http://ai.stanford.edu/~nilsson/MLBOOK.pdf. Accessed on 22 July 2010

Oblinger D (2006) Bootstrapped learning proposer information pamphletfor broad agency announcement 07-04. Online at http://fs1.fbo.gov/EPSData/ODA/Synopses/4965/BAA07-04/BLPIPfinal.pdf

Pratt LY, Mostow J, Kamm CA (1991) Direct transfer of learned information among neural networks. In: Proceedings of the ninth national conference on artificial intelligence (AAAI-91), Anaheim, pp 584–589

Ring M (1991) Incremental development of complex behaviors through automatic construction of sensory-motor hierarchies. In: Proceedings of the eighth international workshop (ML91), San Mateo

Silver D, Mercer R (2002) The task rehearsal method of life-long learning: overcoming impoverished data. In: Cohen R, Spencer B (eds) Advances in artificial intelligence, 15th conference of the Canadian society for computational studies of intelligence (AI 2002), Calgary, 27–29 May 2002. Lecture notes in computer science, vol 2338. Springer, London, pp 90–101

Silver D, Poirier R (2005) Requirements for machine lifelong learning. JSOCS technical report TR-2005-009, Acadia University

Swarup S, Lakkaraju K, Ray SR, Gasser L (2006) Symbol grounding through cumulative learning. In: Vogt P et al (eds) Symbol grounding and beyond: proceedings of the third international workshop on the emergence and evolution of linguistic communication, Rome. Springer, Berlin, pp 180–191

Swarup S, Mahmud MMH, Lakkaraju K, Ray SR (2005) Cumulative learning: towards designing cognitive architectures for artificial agents that have a lifetime. Technical report UIUCDCS-R-2005-2514

Thrun S (1998) Lifelong learning algorithms. In: Thrun S, Pratt LY (eds) Learning to learn. Kluwer Academic, Norwell

Thrun S, Mitchell T (1995) Lifelong robot learning. Robot Auton Syst 15:25–46

Turing AM (1950) Computing machinery and intelligence. Mind Mind 59(236):433–460

Vilalta R, Drissi Y (2002) A perspective view and survey of meta-learning. Artif Intell Rev 18:77–95

Curse of Dimensionality

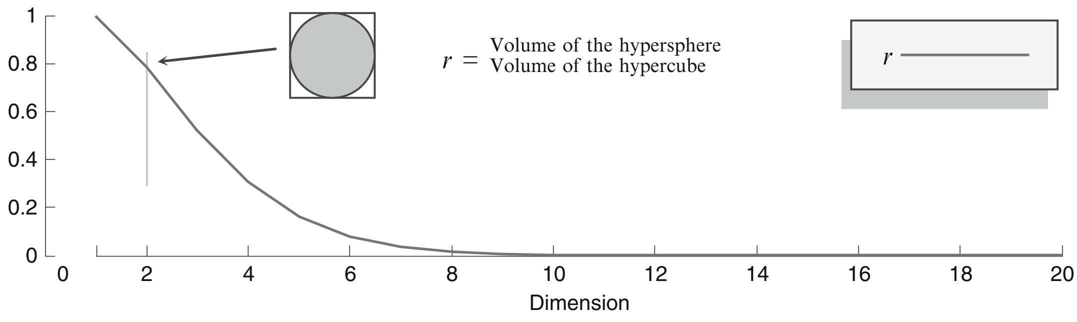
Eamonn Keogh and Abdullah Mueen
University of California-Riverside, Riverside, CA, USA

Definition

The curse of dimensionality is a term introduced by Bellman to describe the problem caused by the exponential increase in volume associated with adding extra dimensions to Euclidean space (Bellman 1957).

For example, 100 evenly-spaced sample points suffice to sample a unit interval with no more than 0.01 distance between points; an equivalent sampling of a 10-dimensional unit hypercube with a grid with a spacing of 0.01 between adjacent points would require 10^{20} sample points: thus, in some sense, the 10D hypercube can be said to be a factor of 10^{18} "larger" than the unit interval.

Informally, the phrase *curse of dimensionality* is often used to simply refer to the fact that one's intuitions about how data structures, sim-



Curse of Dimensionality, Fig. 1 The ratio of the volume of the hypersphere enclosed by the unit hypercube. The most intuitive example, the unit square and unit circle, are shown as an *inset*. Note that the volume of the hypersphere quickly becomes irrelevant for higher dimensionality

ilarity measures, and algorithms behave in low dimensions do typically generalize well to higher dimensions.

Background

Another way to envisage the vastness of high-dimensional Euclidean space is to compare the size of the unit sphere with the unit cube as the dimension of the space increases: as the dimension increases. As we can see in Fig. 1, the unit sphere becomes an insignificant volume relative to that of the unit cube. In other words, almost all of the high-dimensional space is *far away* from the center.

In research papers, the phrase *curse of dimensionality* is often used as shorthand for one of its many implications for machine learning algorithms. Examples of these implications include:

- ▶ Nearest neighbor searches can be made significantly faster for low-dimensional data by indexing the data with an R-tree, a KD-tree, or a similar spatial access method. However, for high-dimensional data all such methods degrade to the performance of a simple linear scan across the data.
- For machine learning problems, a small increase in dimensionality generally requires a large increase in the numerosity of the data, in order to keep the same level of performance for regression, clustering, etc.
- In high-dimensional spaces, the normally intuitive concept of proximity or similarity may not be qualitatively meaningful. This is because the ratio of an object's nearest neighbor over its farthest neighbor approaches one for high-dimensional spaces (Aggarwal 2001). In other words, all objects are approximately equidistant from each other.

There are many ways to attempt to mitigate the *curse of dimensionality*, including ▶ feature selection and ▶ dimensionality reduction. However, there is no single solution to the many difficulties caused by the effect.

Recommended Reading

Aggarwal CC, Hinneburg A, Keim DA (2001) On the surprising behavior of distance metrics in high dimensional spaces. In: ICDT, London, pp 420–434

Bellman RE (1957) Dynamic programming. Princeton University Press, Princeton

Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh E (2008) Querying and mining of time series data: experimental comparison of representations and distance measures. In: Proceedings of the VLDB endowment, Auckland, vol 1, pp 1542–1552

The major database (SIGMOD, VLDB, PODS), data mining (SIGKDD, ICDM, SDM), and machine learning (ICML, NIPS) conferences typically feature several papers which explicitly address the *curse of dimensionality* each year