E

EBL

▸ Explanation-Based Learning

Echo State Network

▸ Reservoir Computing

ECOC

▸ Error Correcting Output Codes

Edge Prediction

▸ Link Prediction

Efficient Exploration in Reinforcement Learning

John Langford
Microsoft Research, New York, NY, USA

Synonyms

PAC-MDP learning

Definition

An agent acting in a world makes observations, takes actions, and receives rewards for the actions taken. Given a history of such interactions, the agent must make the next choice of action so as to maximize the long-term sum of rewards. To do this well, an agent may take suboptimal actions which allow it to gather the information necessary to later take optimal or near-optimal actions with respect to maximizing the long-term sum of rewards. These information gathering actions are generally considered exploration actions.

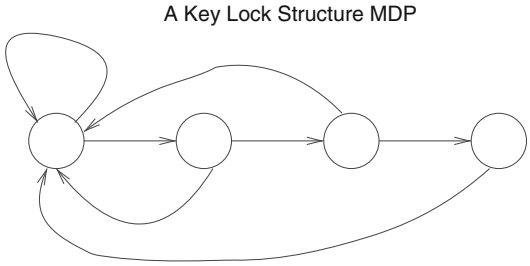
Motivation

Since gathering information about the world generally involves taking suboptimal actions compared with a later learned policy, minimizing the number of information gathering actions helps optimize the standard goal in reinforcement learning. In addition, understanding exploration well is key to understanding reinforcement learning well, since exploration is a key aspect of reinforcement learning which is missing from standard supervised learning settings (Fig. 1).

Efficient Exploration in Markov Decision Processes

One simplification of reinforcement learning is the ▸ Markov decision process setting. In this

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



Efficient Exploration in Reinforcement Learning, Fig. 1 An example of a keylock MDP. The state are arranged in a chain. In each state, one of the two actions leads to the next state while the other leads back to the beginning. The only reward is in the transition to the last state in the chain. Keylock MDPs defeat simple greedy strategies, because the probability of randomly reaching the last transition is exponentially small in the length of the chain

setting, an agent repeatedly takes an action a, resulting in a transition to a state according to a conditional probability transition matrix $P(s'|s,a)$, and a (possibly probabilistic) reward $R(s',a,s) \in [0,1]$. The goal is to efficiently output a policy π which is ε-optimal over T timesteps. The value of policy π in a start state s is defined as

$$\eta(\pi,s) = E_{(a,s,r)^T \sim (\pi,P,R)^T} \sum_{t=1}^{T} r_t,$$

which should be read as the expectation over T-length sequences drawn from the interaction of the policy π with the world as represented by P and R. An ε-optimal policy π therefore satisfies:

$$\max_{\pi'} \eta(\pi',s) - \eta(\pi,s) \leq \varepsilon.$$

There are several notable results in this setting, typically expressed in terms of the dependence on the number of actions A, and the number of states S. The first is for the β-greedy strategy commonly applied when using ▸ Q-learning (Watkins and Dayan 1992) which explores randomly with probability β.

Theorem 1 *There exists MDPs such that with probability at least* $1/2$, β*-greedy requires* $\Theta(A^S)$ *explorations to find an* ε*-optimal policy.*

This is essentially a negative result, saying that a greedy exploration strategy cannot quickly discover a good policy in some settings. The proof uses an MDP with a key-lock like structure where for each state all actions but one take the agent back to the beginning state, and the reward is at the end of a chain of states.

It turns out that there exists algorithms capable of finding a near-optimal policy in an MDP with only a polynomial number of exploratory transitions.

Theorem 2 *For all MDPs, for any* $\delta > 0$, with probability $1-\delta$, the algorithm Explicit-Explore-or-Exploit finds an ε*-optimal policy after* $\tilde{O}(S^2A)$ *explorations.*

In other words, E^3 (Kearns and Singh 1998) requires exploration steps at most proportional to the size of the probability table driving the dynamics of the agent's world. The algorithm works in precisely the manner which might be expected: it builds a model of the world based on its observations and solves the model to determine whether to explore or exploit. The basic approach was generalized to stochastic games and reformulated as an "optimistic initialization" style algorithm named R-MAX (Brafman and Tennenholtz 2002).

It turns out that an even better dependence is possible using the delayed Q-learning (Strehl et al. 2006) algorithm.

Theorem 3 *For all MDPs, for any* $\delta > 0$, with probability $1-\delta$, the algorithm delayed Q-learning finds an ε*-optimal policy after* $\tilde{O}(SA)$ *explorations.*

The delayed Q-learning algorithm requires explorations proportional to the size of the solution policy rather than proportional to the size of world dynamics. At a high level, delayed Q-learning operates by keeping values for exploration and exploitation of observed state-actions, uses these values to decide between exploration and exploitation, and carefully updates these values. Delayed Q-learning does not obsolete E^3, because the (nonvisible) dependence on ε and T are worse (Strehl 2007).

This is a best possible result in terms of the dependence on S and A (up to log factors), as the following theorem (Kakade 2003) states:

Theorem 4 *For all algorithms, there exists an MDP such that with $\Omega(SA)$ explorations are required to find an ε optimal policy with probability at least $\frac{1}{2}$.*

Since even representing a policy requires a lookup table of size SA, this algorithm-independent lower bound is relatively unsurprising.

Variations on MDP Learning

There are several minor variations in the setting and goal definitions which do not qualitatively impact the set of provable results. For example, if rewards are in a bounded range, they can be offset and rescaled to the interval [0, 1].

It's also common to use a soft horizon (or discounting) where the policy evaluation is changed to:

$$\eta_\gamma(\pi, s) = E_{(a,s,r)^\infty \sim (\pi,P,R)^\infty} \sum_{t=1}^{\infty} \gamma^t r_t$$

for some value $\gamma < 1$. This setting is not precisely equivalent to the hard horizon, but since

$$sum_{t=(1n(1/\epsilon)+1n(1/1-\gamma))/1-\gamma}^{\infty} \gamma^t r_t \leq \varepsilon$$

similar results are provable with $1/(1-\gamma)$ taking the role of T and slightly altered algorithms.

One last variation changes the goal. Instead of outputting an ε-optimal policy for the next T timesteps, we could have an algorithm to handle both the exploration and exploitation, then retrospectively go back over a trace of experience and mark a subset of the actions as "exploration actions," with a guarantee that the remainder of the actions are according to an ε-optimal policy (Kakade 2003). Again, minor alterations to known algorithms in the above setting appear to work here.

Alternative Settings

There are several known analyzed variants of the basic setting formed by making additional assumptions about the world. This includes Factored MDPs (Kearns and Koller 1999), Metric MDPs (Kakade et al. 2003), Continuous MDPs (Brunskill et al. 2008), MDPs with a Bayesian prior (Poupart et al. 2006), and apprenticeship learning where there is access to a teacher for an MDP (Abbeel and Ng 2005). The structure of these results are all similar at a high level: with some additional information, it is possible to greatly ease the difficulty of exploration allowing tractable application to much larger problems.

Cross-References

- ▸ *k*-Armed Bandit
- ▸ Reinforcement Learning

Recommended Reading

Abbeel P, Ng A (2005) Exploration and apprenticeship learning in reinforcement learning. In: ICML 2005, Bonn

Brafman RI, Tennenholtz M (2002) R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. J Mach Learn Res 3:213–231

Brunskill E, Leffler BR, Li L, Littman ML, Roy N (2008) CORL: a continuous-state offset-dynamics reinforcement learner. In: UAI-08, Helsinki July 2008

Kakade S (2003) Thesis at gatsby computational neuroscience unit

Kakade S, Kearns M, Langford J (2003) Exploration in metric state spaces. In: ICML 2003, Washington, DC

Kearns M, Koller D (1999) Efficient reinforcement learning in factored MDPs. In: Proceedings of the 16th international joint conference on artificial intelligence. Morgan Kaufmann, San Francisco, pp 740–747

Kearns M, Singh S (1998) Near-optimal reinforcement learning in polynomial time. In: ICML 1998. Morgan Kaufmann, San Francisco, pp 260–268

Poupart P, Vlassis N, Hoey J, Regan K (2006) An analytic solution to discrete Bayesian reinforcement

learning. In: ICML 2006. ACM Press, New York, pp 697–704
Strehl A (2007) Thesis at Rutgers University
Strehl AL, Li L, Wiewiora E, Langford J, Littman ML (2006) PAC model-free reinforcement learning. In: Proceedings of the 23rd international conference on machine learning (ICML 2006), Pittsburgh, pp 881–888
Watkins C, Dayan P (1992) Q-learning. Mach Learn J 8:279–292

EFSC

▸ Evolutionary Feature Selection and Construction

Eigenvector

▸ *K*-Way Spectral Clustering

Elman Network

▸ Simple Recurrent Network

Embodied Evolutionary Learning

▸ Evolutionary Robotics

Emerging Patterns

Definition

Emerging pattern mining is an area of ▸ supervised descriptive rule induction. Emerging patterns are defined as itemsets whose support increases significantly from one data set to another (Dong 1999). Emerging patterns are said to capture emerging trends in time-stamped databases, or to capture differentiating characteristics between classes of data.

Recommended Reading

Dong G, Li J (1999) Efficient mining of emerging patterns: discovering trends and differences. In: Proceedings of the 5th ACM SIGKDD international conference on knowledge discovery and data mining (KDD-99), San Diego, pp 43–52

Empirical Risk Minimization

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

Definition

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

$$P(f(X) \neq Y) = \mathsf{E}_{(X,Y)\sim P_{XY}} [\delta(f(X) \neq Y)],$$

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

However, in practice we only have n pairs of training examples (X_i, Y_i) drawn identically and independently from P_{XY}. Since P_{XY} is unknown, we often use the risk on the training set (called empirical risk) as a surrogate of the expected risk on the underlying distribution:

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

Empirical Risk Minimization (ERM) refers to the idea of choosing a function f by minimizing the empirical risk. Although it is often effective

and efficient, ERM is subject to ▸ overfitting, i.e. finding a model which fits the training data well but predicts poorly on unseen data. Therefore, ▸ regularization is often required.

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

Recommended Reading

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

Ensemble Learning

Gavin Brown
The University of Manchester, Manchester, UK

Synonyms

Committee machines; Multiple classifier systems

Definition

Ensemble learning refers to the procedures employed to train multiple learning machines and combine their outputs, treating them as a "committee" of decision makers. The principle is that the decision of the committee, with individual predictions combined appropriately, should have better overall ▸ accuracy, on average, than any individual committee member. Numerous empirical and theoretical studies have demonstrated that ensemble models very often attain higher accuracy than single models.

The members of the ensemble might be predicting real-valued numbers, class labels, posterior probabilities, rankings, clusterings, or any other quantity. Therefore, their decisions can be combined by many methods, including averaging, voting, and probabilistic methods. The majority of ensemble learning methods are generic, applicable across broad classes of model types and learning tasks.

Motivation and Background

If we could build the "perfect" machine learning device, one which would give us the best possible answer every time, there would be no need for *ensemble learning* methods – indeed, there would be no need for this encyclopedia either. The underlying principle of ensemble learning is a recognition that in real-world situations, every model has limitations and will make errors. Given that each model has these "limitations," the aim of ensemble learning is to manage their strengths and weaknesses, leading to the best possible decision being taken overall. Several theoretical and empirical results have shown that the accuracy of an ensemble can significantly exceed that of a single model.

The principle of combining predictions has been of interest to several fields over many years. Over 200 years ago, a controversial question had arisen, on how best to estimate the mean of a probability distribution given a small number of sample observations. Laplace (1818) demonstrated that the sample mean was not always optimal: under a simple condition, the sample median was a better combined predictor of the population mean. The financial forecasting community has analyzed model combination for several decades, in the context of stock portfolios. The contribution of the machine learning (ML) community emerged in the 1990s – automatic construction (from data) of both the models and the method to combine them. While the majority of the ML literature on this topic is from 1990 onward, the principle has been explored briefly by several independent authors since the 1960s. See Kuncheva (2004b) for historical accounts.

The study of ensemble methods, with model outputs considered for their abstract properties rather than the specifics of the algorithm which produced them, allows for a wide impact across many fields of study. If we can understand precisely why, when, and how particular ensemble methods can be applied successfully, we would have made progress toward a powerful new tool for Machine Learning: *the ability to automatically exploit the strengths and weaknesses of different learning systems*.

Methods and Algorithms

An ensemble consists of a set of models and a method to combine them. We begin this section by assuming that we have a set of models, generated by any of the learning algorithms in this encyclopedia; we explore popular methods of combining their outputs, for classification and regression problems. Following this, we review some of the most popular ensemble algorithms, for *learning* a set of models given the knowledge that they will be combined, including extensive pointers for further reading. Finally, we take a theoretical perspective, and review the concept of ensemble *diversity*, the fundamental property which governs how well an ensemble can perform.

Methods for Combining a Set of Models

There exist numerous methods for model combination, far too many to fully detail here. The *linear* combiner, the *product* combiner, and the *voting* combiner are by far the most commonly used in practice. Though a combiner could be specifically chosen to optimize performance in a particular application, these three rules have shown consistently good behavior across many problems, and are simple enough that they are amenable to theoretical analysis.

The linear combiner is used for models that output real-valued numbers, so is applicable for ▸ regression ensembles, or for ▸ classification ensembles producing class probability estimates. Here, notation for the latter case is only shown. We have a model $f_t(y|\mathbf{x})$, an estimate of the probability of class y given input $\mathbf{x}$. For a set of these, $t = \{1, \ldots, \mathrm{T}\}$, the ensemble probability estimate is,

$$\bar{f}(y|\mathbf{x}) = \sum_{t=1}^{T} w_t f_t(y|\mathbf{x}). \qquad (1)$$

If the weights $w_t = 1/T$, $\forall t$, this is a simple uniform averaging of the probability estimates. The notation clearly allows for the possibility of a nonuniformly weighted average. If the classifiers have different accuracies on the data, a nonuniform combination could *in theory* give a lower error than a uniform combination. However, in practice, the difficulty of estimating the $\mathbf{w}$ parameters without overfitting, and the relatively small gain that is available (see Kuncheva 2004b, p. 282), have meant that in practice the uniformly weighted average is by far the most commonly used. A notable exception, to be discussed later in this article, is the *mixture of experts* paradigm – in MoE, weights are nonuniform, but are learnt and dependent on the input value $\mathbf{x}$. An alternative combiner is the *product rule*:

$$\bar{f}(y|\mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} f_t(y|\mathbf{x})^{wt}, \qquad (2)$$

where Z is a normalization factor to ensure $\bar{f}$ is a valid distribution. Note that Z is not *required* to make a valid decision, as the order of posterior estimates remain unchanged before/after normalization. Under the assumption that the class-conditional probability estimates are independent, this is the theoretically optimal combination strategy. However, this assumption is highly unlikely to hold in practice, and again the weights $\mathbf{w}$ are difficult to reliably determine. Interestingly, the linear and product combiners are in fact special cases of the *generalized mean* (Kuncheva 2004b) allowing for a continuum of possible combining strategies.

The linear and product combiners are applicable when our models output real-valued numbers. When the models instead output class labels, a majority (or plurality) vote can be used. Here, each classifier votes for a particular class, and the class with the most votes is chosen as the ensemble output. For a two-class problem the models produce labels, $h_t(\mathbf{x}) \in \{-1, +1\}$. In this case, the ensemble output for the *voting* combiner can be written as

$$H(\mathbf{x}) = sign\left(\sum_{t=1}^{T} w_t h_t(x)\right). \qquad (3)$$

The weights $\mathbf{w}$ can be uniform for a simple majority vote, or nonuniform for a weighted vote.

We have discussed only a small fraction of the possible combiner rules. Numerous other rules exist, including methods for combining rankings of classes, and unsupervised methods to combine clustering results. For details of the wider literature, see Kuncheva (2004b) or Polikar (2006).

Algorithms for Learning a Set of Models

If we had a committee of *people* taking decisions, it is self-evident that we would not want them all to make the same bad judgments *at the same time*. With a committee of learning models, the same intuition applies: we will have no gain from combining a set of identical models. We wish the models to exhibit a certain element of "diversity" in their group behavior, though still retaining good performance individually.

We therefore make a distinction between two types of ensemble learning algorithms, those which encourage diversity *implicitly*, and those which encourage it *explicitly*. The vast majority of ensemble methods are *implicit*, in that they provide different *random subsets* of the training data to each learner. Diversity is encouraged "implicitly" by *random* sampling of the data space: at no point is a *measurement* taken to ensure diversity will emerge. The random differences between the datasets might be in the selection of examples (the ▸ Bagging algorithm), the selection of features (▸ Random Subspace Method, Ho (1998) or ▸ Rotation Forests, Rodriguez et al. 2006), or combinations of the two (the Random Forests algorithm, Breiman 2001). Many other "randomization" schemes are of course possible.

An alternative is to *explicitly* encourage diversity, constructing each ensemble member with some measurement ensuring that it is substantially different from the other members. ▸ Boosting algorithms achieve this by altering the distribution of training examples for each learner such that it is encouraged to make more accurate predictions where previous predictors have made errors. The DECORATE algorithm (Melville and Mooney 2005) explicitly alters the distribution of class labels, such that successive models are forced to learn different answers to the same problem. ▸ Negative correlation learning (see Brown 2004; Brown et al. 2005), includes a penalty term when learning each ensemble member, explicitly *managing* the accuracy-diversity trade-off.

In general, ensemble methods constitute a large class of algorithms – some based on heuristics, and some on sound learning-theoretic principles. The three algorithms that have received the most attention in the literature are reviewed here. It should be noted that we present only the most basic form of each; numerous modifications have been proposed for a variety of learning scenarios. As further study the reader is referred to the many comprehensive surveys of the field (Brown et al. 2005; Kuncheva 2004b; Polikar 2006).

Bagging

In the Bagging algorithm (Breiman 1996) each member of the ensemble is constructed from a different training dataset, and the predictions combined either by uniform averaging or voting over class labels. Each dataset is generated by sampling from the total N data examples, choosing N items uniformly at random *with replacement*. Each sample is known as a *bootstrap*; the name Bagging is an acronym derived from *B*ootstrap *AGG*regat*ING*. Since a bootstrap samples N items uniformly at random with replacement, the probability of any individual data item *not* being selected is $p = (1-1/N)^N$. Therefore with large N, a single bootstrap is expected to contain approximately 63. 2 % of the original set, while 36. 8 % of the originals are not selected.

Like many ensemble methods, Bagging works best with *unstable* models, that is those that produce differing generalization behavior with small changes to the training data. These are also known as *high variance* models, examples of which are ▸ decision trees and ▸ neural networks. Bagging therefore tends not to work well with very simple models. In effect, Bagging samples randomly from the space of possible models to make up the ensemble – with very simple models the sampling produces almost identical (low diversity) predictions.

Despite its apparent capability for variance reduction, situations have been demonstrated where

Algorithm 1 Bagging

Input: Required ensemble size T
Input: Training set $S = \{(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)\}$
for $t = 1$ to T **do**
Build a dataset S_t, by sampling N items, randomly *with replacement* from S.
Train a model h_t using S_t, and add it to the ensemble
end for
For a new testing point (x', y'),
If model outputs are continuous, combine them by voting.

Bagging can converge *without* affecting variance (see Brown et al. 2005). Several other explanations have been proposed for Bagging's success, including links to Bayesian model averaging. In summary, it seems that several years from its introduction, despite its apparent simplicity, Bagging is still not fully understood.

Adaboost

Adaboost (Freund and Schapire 1996) is the most well known of the *Boosting* family of algorithms (Schapire 2003). The algorithm trains models sequentially, with a new model trained at each round. At the end of each round, mis-classified examples are identified and have their emphasis increased in a new training set which is then fed back into the start of the next round, and a new model is trained. The idea is that subsequent models should be able to compensate for errors made by earlier models.

Adaboost occupies somewhat of a special place in the history of ensemble methods. Though the procedure seems heuristic, the algorithm is in fact grounded in a rich learning-theoretic body of literature. (Schapire 1990) addressed a question posed by Kearns and Valiant (1988) on the nature of two complexity classes of learning problems. The two classes are *strongly learnable* and *weakly learnable* problems. Schapire showed that these classes were equivalent; this had the corollary that a weak model, performing only slightly better than random guessing, could be "boosted" into an arbitrarily accurate *strong* model. The original Boosting algorithm was a proof by construction of this equivalence, though had a number of impractical assumptions built-in. The Adaboost algorithm (Freund and Schapire 1996) was the first practical Boosting method. The authoritative historical account of the development can be found in Schapire (1999), including discussion of numerous variants and interpretations of the algorithm. The procedure is shown in Algorithm 2. Some similarities with Bagging are evident; a key differences is that at each round t, Bagging has a uniform distribution D_t, while Adaboost adapts a nonuniform distribution.

Algorithm 2 Adaboost

Input: Required ensemble size T
Input: Training set $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where $y_j \in \{-1, +1\}$
Define a uniform distribution $D_1(i)$ over elements of S.
for $t = 1$ to T **do**
Train a model h_t using distribution D_t.
Calculate $e_t = P_{D_t}(h_t(x) \neq y)$
If $\epsilon_t \geq 0.5$ break
Set $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
Update $D_{t+1}(i) = \dfrac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
where Z_t is a normalization factor so that D_{t+1} is a valid distribution.
end for
For a new testing point (x', y') ,
$H(x') = \text{sign}(\Sigma_{t=1}^{T} \alpha_t h_t(x'))$

The ensemble is constructed by iteratively adding models. Each time a model is learnt, it is checked to ensure it has at least $\varepsilon_t < 0.5$, that is, it has performance *better than random guessing* on the data it was supplied with. If it does not, either an alternative model is constructed, or the loop is terminated.

After each round, the distribution D_t is updated to emphasize incorrectly classified examples. The update causes half the distribution mass of D_{t+1} to be over the examples incorrectly classified by the previous model. More precisely, $\sum_{h_t(x_1) \neq y_i} D_{t+1}(i) = 0.5$. Thus, if h_t has an error rate of 10 %, then examples from that small 10 % will be allocated 50 % of the next model's training "effort," while the remaining examples

(those correctly classified) are underemphasized. An equivalent (and simpler) writing of the distribution update scheme is to multiply $D_t(i)$ by $1/2(1-\varepsilon_t)$ if $h_t(x_i)$ is correct, and by $1/2\varepsilon_t$ otherwise.

The updates cause the models to sequentially minimize an exponential bound on the error rate. The training error rate on a data sample $\mathcal{S}$ drawn from the true distribution $\mathcal{D}$ obeys the bound,

$$P_{x,y\sim S}(yH(x) < 0) \leq \prod_{t=1}^{T} 2\sqrt{\varepsilon_t(1-\varepsilon_t)}. \quad (4)$$

This *upper bound* on the training error (though not the *actual* training error) is guaranteed to decrease monotonically with T, given $\varepsilon_t < 0.5$.

In an attempt to further explain the performance of Boosting algorithms, Schapire also developed bounds on the *generalization* error of voting systems, in terms of the voting margin, the definition of which was given in (10). Note that, this is not the same as the *geometric margin*, optimized by ▸ support vector machines. The difference is that the voting margin is defined using the one-norm $||\mathbf{w}||_1$ in the denominator, while the geometric margin uses the *two*-norm $||\mathbf{w}||_2$. While this is a subtle difference, it is an important one, forming links between SVMs and Boosting algorithms – see Rätsch et al. (2002) for details. The following bound holds with probability $1-\delta$,

$$P\,x, y \sim \mathcal{D}(H(x) \neq y) \leq P_{x,y\sim\mathcal{S}}(yH(x) < \theta) + \tilde{O}\left(\sqrt{\frac{d}{N\theta^2} - ln\delta}\right), \quad (5)$$

where the $\tilde{O}$ notation hides constants and logarithmic terms, and d is the ▸ VC-dimension of the model used. Roughly, this states that the generalization error is less than or equal to the training error plus a term dependent on the voting margin. The larger the minimum margin in the training data, the lower the testing error. The original bounds have since been significantly improved, see Koltchinskii and Panchenko (2005) as a comprehensive recent work. We note that this bound holds generally for *any* voting system, and is not specific to the Boosting framework.

The margin-based theory is only one explanation of the success of Boosting algorithms. Mease and Wyner (2008) present a discussion of several questions on why and how Adaboost succeeds. The subsequent 70 pages of discussion demonstrate that the story is by no means simple. The conclusion is, while no single theory can fully explain Boosting, each provides a different part of the still unfolding story.

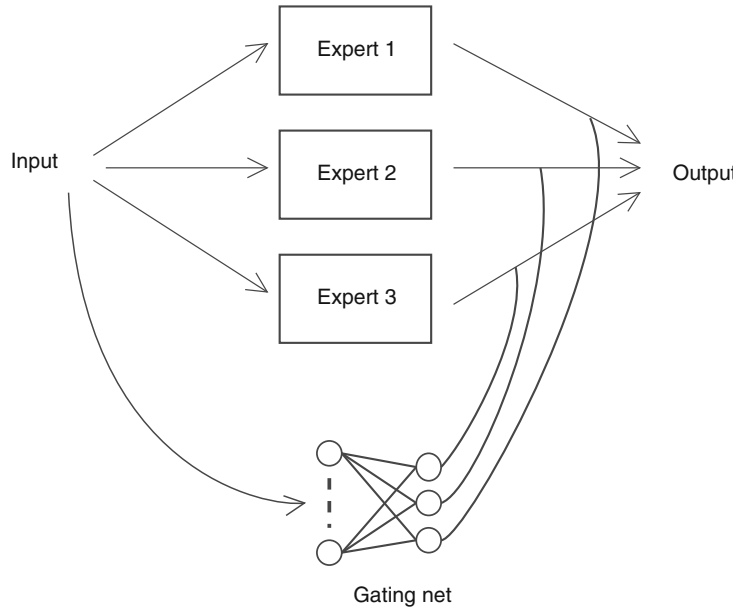
Mixtures of Experts

The mixtures of experts architecture is a widely investigated paradigm for creating a combination of models (Jacobs et al. 1991). The principle underlying the architecture is that certain models will be able to "specialize" to particular parts of the input space. It is commonly implemented with a neural network as the base model, or some other model capable of estimating probabilities. A *Gating network* receives the same inputs as the component models, but its outputs are used as the weights for a linear combiner. The Gating network is responsible for learning the appropriate weighted combination of the specialized models ("experts") for any given input. Thus, the input space is "carved-up" between the experts, increasing and decreasing their weights for particular examples. In effect, a mixture of experts explicitly learns how to create expert ensemble members in different portions of the input space, and select the most appropriate subset for a new testing example (Fig. 1).

The architecture has received wide attention, and has a strong following in the probabilistic modeling community, where it may go under the pseudonym of a "mixture model." A common training method is the ▸ expectation-maximization algorithm.

Theoretical Perspectives: Ensemble Diversity

We have seen that all ensemble algorithms in some way attempt to encourage "diversity." In



Ensemble Learning, Fig. 1 The mixtures of experts architecture

this section, we take a more formalized perspective, to understand what is meant by this term.

What Is Diversity?

The optimal "diversity" is fundamentally a *credit assignment* problem. If the committee as a whole makes an erroneous prediction, how much of this error should be attributed to each member? More precisely, how much of the committee prediction is due to the accuracies of the individual models, and how much is due to their interactions when they were combined? We would ideally like to reexpress the ensemble error as two distinct components: a term for the accuracies of the individual models, plus a term for their interactions, i.e., their *diversity*.

It turns out that this so-called *accuracy-diversity* breakdown of the ensemble error is not always possible, depending on the type of error function, and choice of combiner rule. It should be noted that when "diversity" is referred to in the literature, it is most often meant to indicate classification with a majority vote combiner, but for completeness we address the general case here. In the following sections, the existing work to understand diversity in three distinct cases is described: for regression tasks (a linear combiner), and classification tasks, with either a linear combiner or a voting combiner.

Regression Error with a Linear Combination Rule

In a regression problem, it is common to use the squared error criterion. The accuracy-diversity breakdown for this case (using a linear combiner) is called the *ambiguity decomposition* (Krogh and Vedelsby 1995). The result states that the squared error of the linearly combined ensemble, $\bar{f}(\mathbf{x})$, can be broken into a sum of two components:

$$(\bar{f}(\mathbf{x}) - d)^2 = \frac{1}{T}\sum_{t=1}^{T}(f_t(\mathbf{x}) - d)^2 - \frac{1}{T}\sum_{t=1}^{T}(f_t(\mathbf{x}) - \bar{f}(\mathbf{x}))^2. \quad (6)$$

The first term on the right hand side is the average squared error of the individual models, while the second term quantifies the interactions *between* the predictions. Note that this second term, the "ambiguity," is always positive. This guarantees that, for an arbitrary data point, the ensemble squared error is always less than or equal to the average of the individual squared errors.

The intuition here can be understood as follows. Imagine five friends, playing "guess the weight of the cake" (an old English fairground game): if a player's guess is close enough to the true weight, they win the cake. Just as they are about to play, the fairground manager states that they can only submit *one* guess. The dilemma seems to be in whose guess they should submit – however, the ambiguity decomposition shows us that taking the average of their guesses, and submitting that, will *always* be closer (on average) than choosing a person at random and submitting their guess. Note that this is qualified with "on average" – it may well be that one of the predictions will in fact be closer than the average prediction, but we presume that we have no way of identifying *which* prediction to choose, other than random. It can be seen that greater diversity in the predictions (i.e., a larger ambiguity term) results in a larger gain over the average individual performance. However, it is also clear that there is a trade-off to be had: too much diversity and the average error is extremely large.

The idea of a trade-off between these two terms is reminiscent of the ▸ bias-variance decomposition (Geman et al. 1992); in fact, there is a deep connection between these results. Taking the expected value of (6) over all possible training sets gives us the ensemble analogy to the bias-variance decomposition, called the ▸ bias-variance-covariance decomposition (Ueda and Nakano 1996). This shows that the expected squared error of an ensemble $\bar{f}(\mathbf{x})$ from a target d is:

$$\mathcal{E}_{\mathcal{D}}\{(\bar{f}(\mathbf{x}) - d)^2\} = \overline{\text{bias}}^2 + \frac{1}{T}\overline{\text{var}} + \left(1 - \frac{1}{T}\right)\overline{\text{covar}}, \quad (7)$$

where the expectation is with respect to all possible training datasets $\mathcal{D}$. While the bias and variance terms are constrained to be positive, the covariance between models can become negative – thus the definition of diversity emerges as an extra degree of freedom in the bias-variance dilemma. This extra degree of freedom allows an ensemble to approximate functions that are difficult (if not impossible) to find with a single model. See Brown et al. (2005) for extensive further discussion of this concept.

Classification Error with a Linear Combination Rule

In a classification problem, our error criterion is the misclassification rate, also known as the *zero-one* loss function. For this type of loss, it is well known there is no unique definition of bias-variance; instead there exist multiple decompositions each with advantages and disadvantages (see Kuncheva 2004b, p. 224). This gives us a clue as to the situation with an ensemble – there is also no simple accuracy-diversity separation of the ensemble classification error. Classification problems can of course be addressed either by a model producing class probabilities (where we linearly combine), or directly producing class labels (where we use majority vote). Partial theory has been developed for each case.

For linear combiners, there exist theoretical results that relate the correlation of the probability estimates to the ensemble classification error. Tumer and Ghosh (1996) showed that the reducible classification error (i.e., above the Bayes rate) of a simple averaging ensemble, e_{ave}, can be written as

$$e_{\text{ave}} = e_{\text{add}}\left(\frac{1 + \delta(T-1)}{T}\right), \quad (8)$$

where e_{add} is the classification error of an individual model. The δ is a correlation coefficient between the model outputs. When the individual models are identical, the correlation is $\delta = 1$. In this case, the ensemble error is equal to the individual error, $e_{\text{ave}} = e_{\text{add}}$. When the models are statistically independent, $\delta = 0$, and the ensemble error is a fraction $1/T$ of the individual error, $e_{\text{ave}} = 1/T \times \text{e}_{\text{add}}$. When δ is negative, the models are negatively correlated, and the ensemble error is lower than the average individual error. However, (8) is derived under quite strict assumptions, holding only for a local area around the decision boundary, and ultimately resting on

the bias-variance-covariance theory from regression problems. Further details, including recent work to lift some of the assumptions (Kuncheva 2004b).

Classification Error with a Voting Combination Rule

The case of a classification problem with a majority vote combiner is the most challenging of all. In general, there is no known breakdown of the ensemble classification error into neat accuracy and diversity components. The simplest intuition to show that correlation between models does affect performance is given by the Binomial theorem. If we have T models each with identical error probability $p = P(h_t(\mathbf{x}) \neq y)$, assuming they make statistically *independent* errors, the following error probability of the majority voting committee holds,

$$P(H(x) \neq y) = \sum_{k>T/2}^{T} \binom{T}{k} p^k (1-p)^{T-k}. \tag{9}$$

For example, in the case of $T = 21$ ensemble members, each with error $p = 0.3$, the majority voting error will be 0. 026, an order of magnitude improvement over the individual error. However, this *only* holds for statistically independent errors. The correlated case is an open problem. Instead, various authors have proposed their own heuristic definitions of diversity in majority voting ensembles. Kuncheva (2004b) conducted extensive studies of several suggested diversity measures; the conclusion was that "*no measure consistently correlates well with the majority vote accuracy.*" In spite of this, some were found useful as an approximate guide to characterize performance of ensemble methods, though should not be relied upon as the "final word" on diversity. Kuncheva's recommendation in this case is the *Q-statistic* (Kuncheva 2004b, p. 299), due to its simplicity and ease of computation.

Breiman (2001) took an alternative approach, deriving not a *separation* of error components, but a *bound* on the generalization error of a voting ensemble, expressed in terms of the correlations of the models. To understand this, we must introduce concept of *voting margin*. The voting margin for a two-class problem, with $y \in \{-1, +1\}$, is defined,

$$m = \frac{y_t \sum_{t=1}^{T} w_t h_t(\mathbf{x})}{\sum_{t=1}^{T} |w_t|} = yH(\mathbf{x}). \tag{10}$$

If the margin is positive, the example is correctly classified, if it is negative, the example is incorrectly classified. The expected margin $s = \mathcal{E}_{\mathcal{D}}\{m\}$ measures the extent to which the average number of votes for the correct class exceeds the average vote for any other class, with respect to the data distribution $\mathcal{D}$. The larger the voting margin, the more confidence in the classification. Breiman's bound shows,

$$P_{\mathcal{D}}(H(\mathbf{x}) \neq y) = P_{\mathcal{D}}(yH(\mathbf{x}) < 0) \neq \frac{\bar{\rho}(1-s^2)}{s^2}. \tag{11}$$

Here $\bar{\rho}$ is the average pairwise correlation between the errors of the individual models. Thus, the generalization error is minimized by a small $\bar{\rho}$, and an s as close to 1 as possible. The balance between a high accuracy (large s) and a high diversity (low $\bar{\rho}$) constitutes the tradeoff in this case, although the bound is quite loose.

Summary

In summary, the definition of diversity depends on the problem. In a regression problem, the optimal diversity is the trade-off between the bias, variance and covariance components of the squared error. In a classification problem, with a linear combiner, there exists partial theory to relate the classifier correlations to the ensemble error rate. In a classification problem with a voting combiner, there is no single theoretical framework or definition of diversity. However, the lack of an agreed definition of diversity has not discouraged researchers from trying to achieve it, nor has it stalled the progress of effective algorithms in the field.

Conclusions and Current Directions in the Field

Ensemble methods constitute some of the most robust and accurate learning algorithms of the past decade (Caruana and Niculescu-Mizil 2006). A multitude of heuristics have been developed for randomizing the ensemble parameters, to generate diverse models. It is arguable that this line of investigation is nowadays rather oversubscribed, and the more interesting research is now in methods for nonstandard data. ▸ Cluster ensembles (Strehl and Ghosh 2003) are ensemble techniques applied to unsupervised learning problems. Problems with *nonstationary* data, also known as *concept drift*, are receiving much recent attention (Kuncheva 2004a). The most up to date innovations are to be found in the biennial *International Workshop on Multiple Classifier Systems* (Roli et al. 2000).

Recommended Reading

Kuncheva (2004b) is the standard reference in the field, which includes references to many further recommended readings. In addition, Brown et al. (2005) and Polikar (2006) provide extensive literature surveys. Roli et al. (2000) is an international workshop series dedicated to ensemble learning.

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Brown G (2004) Diversity in neural network ensembles. PhD thesis, University of Birmingham

Brown G, Wyatt JL, Harris R, Yao X (2005) Diversity creation methods: a survey and categorisation. J Inf Fusion 6(1):5–20

Caruana R, Niculescu-Mizil A (2006) An empirical comparison of supervised learning algorithms. In: Proceedings of the 23rd international conference on machine learning. ACM, New York, pp 161–168

Freund Y, Schapire R (1996) Experiments with a new boosting algorithm. In: Proceedings of the thirteenth international conference on machine learning (ICML'96). Morgan Kauffman Publishers, San Francisco, pp 148–156

Geman S, Bienenstock E, Doursat R (1992) Neural networks and the bias/variance dilemma. Neural Comput 4(1):1–58

Ho TK (1998) The random subspace method for constructing decision forests. IEEE Trans Pattern Anal Mach Intell 20(8):832–844

Jacobs RA, Jordan MI, Nowlan SJ, Hinton GE (1991) Adaptive mixtures of local experts. Neural Comput 3(1):79–87

Kearns M, Valiant LG (1988) Learning Boolean formulae or finite automata is as hard as factoring. Technical report TR-14-88, Harvard University Aiken Computation Laboratory

Koltchinskii V, Panchenko D (2005) Complexities of convex combinations and bounding the generalization error in classification. Ann Stat 33(4):1455

Krogh A, Vedelsby J (1995) Neural network ensembles, crossvalidation and active learning. In: Advances in neural information processing systems. MIT Press, Cambridge, pp 231–238

Kuncheva LI (2004a) Classifier ensembles for changing environments. In: International workshop on multiple classifier systems. Lecture notes in computer science, vol 3007. Springer, Berlin

Kuncheva LI (2004b) Combining pattern classifiers: methods and algorithms. Wiley, New York

Laplace PS (1818) Deuxieme supplement a la theorie analytique des probabilites. Gauthier-Villars, Paris

Mease D, Wyner A (2008) Evidence contrary to the statistical view of Boosting. J Mach Learn Res 9:131–156

Melville P, Mooney RJ (2005) Creating diversity in ensembles using artificial data. Inf Fusion 6(1):99–111

Polikar R (2006) Ensemble based systems in decision making. IEEE Circ Syst Mag 6(3):21–45

Rätsch G, Mika S, Schölkopf B, Müller KR (2002) Constructing Boosting algorithms from SVMs: an application to one-class classification. IEEE Trans Pattern Anal Mach Intell 24(9):1184–1199

Rodriguez J, Kuncheva L, Alonso C (2006) Rotation forest: a new classifier ensemble method. IEEE Trans Pattern Anal Mach Intell 28(10):1619–1630

Roli F, Kittler J, Windridge D, Oza N, Polikar R, Haindl M et al (eds) Proceedings of the international workshop on multiple classifier systems 2000–2009. Lecture notes in computer science. Springer, Berlin. Available at: http://www.informatik.uni-trier.de/ley/db/conf/mcs/index.html

Schapire RE (1990) The strength of weak learnability. Mach Learn 5:197–227

Schapire RE (1999) A brief introduction to boosting. In: Proceedings of the 16th international joint conference on artificial intelligence. Morgan Kaufmann, San Francisco, pp 1401–1406

Schapire RE (2003) The boosting approach to machine learning: an overview. In: Denison DD, Hansen MH, Holmes C, Mallick B, Yu B (eds) Nonlinear estimation & classification Lecture notes in statistics. Springer, Berlin, pp 149–172

Strehl A, Ghosh J (2003) Cluster ensembles – a knowledge reuse framework for combining multiple partitions. J Mach Learn Res 3:583–617

Tumer K, Ghosh J (1996) Error correlation and error reduction in ensemble classifiers. Connect Sci 8(3–4):385–403
Ueda N, Nakano R (1996) Generalization error of ensemble estimators. In: Proceedings of IEEE international conference on neural networks, vol 1, pp 90–95. ISBN:0-7803-3210-5

Entailment

Synonyms

Implication; Logical consequence

Definition

The term entailment is used in the context of logical reasoning. Formally, a logical formula T entails a formula c if and only if all models of T are also a model of c. This is usually denoted as $T \models c$ and means that c is a logical consequence of T or that c is implied by T.

Let us elaborate this definition for propositional clausal logic, where the formulae T could be the following expression:

```
flies :- bird, normal.
bird :- blackbird.
bird :- ostrich.
```

Here, the first clause or rule can be read as flies *if* normal *and* bird, that is, normal birds fly, the second and third one as stating that blackbirds, resp. ostriches, are birds. An interpretation is then an assignment of truth-values to the propositional variables. For instance, for the above domain

```
{ostrich, bird}
{blackbird, bird, normal}
```

are interpretations, specified through the set of propositional variables that are true. This means that in the first interpretation, the only true propositions are `ostrich` and `bird`. An interpretation specifies a kind of possible world. An interpretation I is then a model for a clause $h :- b_1, \ldots b_n$ if and only if $\{b_1, \ldots, b_n\} \subseteq I \rightarrow h \in I$ and it is model for a clausal theory if and only if it is a model for all clauses in the theory. Therefore, the first interpretation above is a model for the theory, but the second one is not because the interpretation is not a model for the first clause (as $\{$`bird, normal`$\} \subseteq I$ but `flies` $\notin I$). Using these notions, it can now be verified that the clausal theory T above logically entails the clause

```
flies :- ostrich, normal.
```

because all models of the theory are also a model for this clause.

In machine learning, the notion of entailment is used as a covers relation in ▸ inductive logic programming, where hypotheses are clausal theories, instances are clauses, and an example is covered by the hypothesis when it is entailed by the hypothesis.

Cross-References

▸ Inverse Entailment
▸ Logic of Generality

Recommended Reading

Russell S, Norvig P (1995) Artificial intelligence: a modern approach, 2nd edn. Prentice Hall, Englewood Cliffs

Entity Resolution

Indrajit Bhattacharya[1] and Lise Getoor[2]
[1]IBM India Research Laboratory, New Delhi, India
[2]University of Maryland, College Park, MD, USA

Abstract

References to real-world entities are often ambiguous, more commonly across data sources but frequently within a single data source as well. Ambiguities occur due to multiple reasons, such as incorrect data entry, or multiple possible representations of the entities. Given such a collection of ambiguous entity references, the goal of entity resolution

is to discover the unique set of underlying entities, and map each reference to its corresponding entity. Resolving such entity ambiguities is necessary for removing redundancy and also for accurate entity-level analysis. This is a common problem that comes up in many different applications and has been studied in different branches of computer science. As evidences for entity resolution, traditional approaches consider pair-wise similarity between references, and many sophisticated similarity measures have been proposed to compare attributes of references. The simplest solution classifies reference pairs with similarity above a threshold as referring to the same entity. More sophisticated solutions use a probabilistic framework for reasoning with the pair-wise probabilities. Recently proposed relational approaches for entity resolution make use of relationships between references when available as additional evidences. Instead of reasoning independently for each pair of references, these approaches reason collectively over related pair-wise decisions over references. One line of work within the relational family uses supervised or unsupervised probabilistic learning using probabilistic graphical models, while another uses more scalable greedy techniques for merging references in a hyper-graph. Beyond improving entity resolution accuracy, such relational approaches yield additional knowledge in the form of relationships between the underlying entities.

Synonyms

Co-reference resolution; Deduplication; Duplicate detection; Identity uncertainty; Merge-purge; Object consolidation; Record linkage; Reference reconciliation

Definition

A fundamental problem in data cleaning and integration (see ▸ Data Preparation) is dealing with uncertain and imprecise references to real-world entities. The goal of entity resolution is to take a collection of uncertain entity references (or references, in short) from a single data source or multiple data sources, discover the unique set of underlying entities, and map each reference to its corresponding entity. This typically involves two subproblems – identification of references with different attributes to the same entity and disambiguation of references with identical attributes by assigning them to different entities.

Motivation and Background

Entity resolution is a common problem that comes up in different guises (and is given different names) in many computer science domains. Examples include computer vision, where we need to figure out when regions in two different images refer to the same underlying object (the correspondence problem), natural language processing when we would like to determine which noun phrases refer to the same underlying entity (co-reference resolution), and databases, where, when merging two databases or cleaning a database, we would like to determine when two tuple records are referring to the same real-world object (deduplication and data integration). Deduplication is important for removing redundancy and for accurate analysis. In information integration, determining approximate joins is important for consolidating information from multiple sources; most often there will not be a unique key that can be used to join tables across databases.

Such ambiguities in entity references can occur due to multiple reasons. Often times, data may have data entry errors, such as typographical errors. Multiple representations, such as abbreviations, are also possible. Different databases typically have different keys – one person database may use social security numbers, while another uses name and address.

Traditional entity resolution approaches focus on matching attributes of different references for resolving entities. However, many data sources have explicit or implicit relationships present among the entity references. These relations

are indicative of relationships between the underlying entities themselves. For example, person records in census data are linked by family relationships such as sibling, parent, and spouse. Researchers collaborate mostly within their organization, or their research community, as a result of which references to related researchers tend to occur closely together. Recent entity resolution approaches in statistical relational learning make use of relationships between references to improve entity resolution accuracy and additionally to discover relationships between the underlying entities.

Theory/Solution

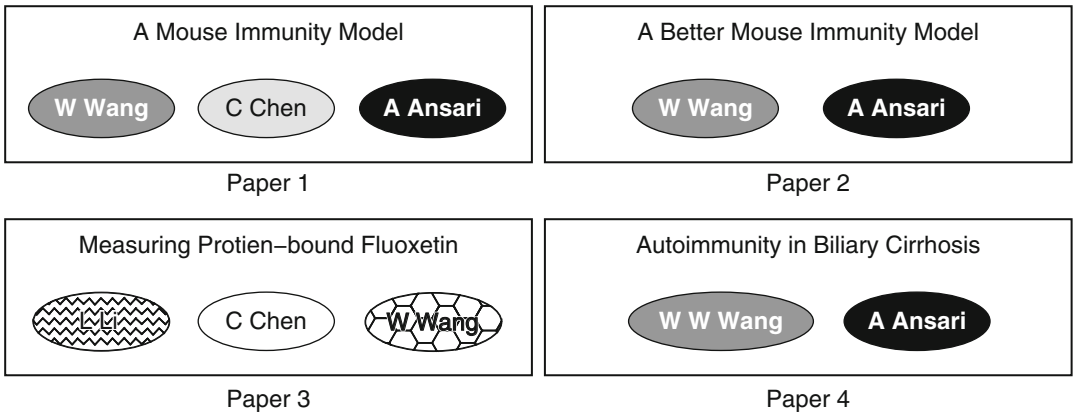
As an illustration of the entity resolution problem, consider the task of resolving the author references in a database of academic publications similar to DBLP, CiteSeer, or PubMed. Let us take as an example the following set of four papers:

1. W. Wang, C. Chen, and A. Ansari, "A mouse immunity model"
2. W. Wang and A. Ansari, "A better mouse immunity model"
3. L. Li, C. Chen, and W. Wang, "Measuring protein-bound fluoxetin"
4. W. W. Wang and A. Ansari, "Autoimmunity in biliary cirrhosis"

Now imagine that we would like to find out, given these four papers, which of these author names refer to the same author entities. This process involves determining whether paper 1 and paper 2 are written by the same author named Wang, or whether they are different authors. We need to answer similar questions about all such similar author names in the database.

In this example, it turns out there are six underlying author entities, which we will call *Wang*1 and *Wang*2, *Chen*1 and *Chen*2, *Ansari*, and *Li*. The three references with the name "A. Ansari" correspond to author $Ansari$ and the reference with name "L. Li" to author *Li*. However, the two references with name "C. Chen" map to two different authors *Chen*1 and *Chen*2. Similarly, the four references with name "W. Wang" or "W. W. Wang" map to two different authors. The "Wang" references from the first, second, and fourth papers correspond to author *Wang*1, while that from the third paper maps to a different author *Wang*2. This inference illustrates the twin problems of *identifying* "W. Wang" and "W. W. Wang" as the same author and *disambiguating* two references with name "W. Wang" as different authors. This is shown pictorially in Fig. 1, where references that correspond to the same authors are shaded identically. In the entity resolution process, all those and only those author references that are shaded identically should be resolved as corresponding to the same underlying entity.

Formally, in the entity resolution problem, we are given a set of references $\mathcal{R} = \{r_i\}$, where each reference r has attributes $r.A_1, r.A_2, \ldots, r.A_k$, such as observed names and affiliations for author references, as in our example above. The references correspond to some set of unknown entities $\mathcal{E} = \{e_i\}$. We introduce the notation $r.E$ to refer to the entity to which reference r corresponds. The goal is to recover the hidden set of entities $\mathcal{E} = \{e_i\}$ and the entity labels $r.E$ for individual references given the observed attributes of the references. In addition to the attributes, in some data sources we have information in the form of relationships between the references, such as coauthor relationships between author references in publication databases. We can capture the relationships with a set of hyper-edges $\mathcal{H} = \{h_i\}$. Each hyper-edge h may have attributes as well to capture the attributes of relationships, which we denote $h.A_1, h.A_2, \ldots, h.A_l$, and we use $h.R$ to denote the set of references that it connects. In our example, each rectangle denotes one hyper-edge corresponding to one paper in the database. The first hyper-edge corresponding to $Paper1$ has as its attribute the title "A mouse immunity model" and connects the three references having name attributes "W. Wang," "C. Chen," and "A. Ansari." A reference r can belong to zero or more hyper-edges, and we use $r.H$ to denote the set of hyper-edges in which r participates. For example, if we have paper, author, and venue references,



Entity Resolution, Fig. 1 The references in different papers in the bibliographic example. References to the same entity are identically shaded

then a paper reference may be connected to multiple author references and also to a venue reference. In general, the underlying references can refer to entities of different types, as in a publication database or in newspaper articles, which contain references to people, places, organizations, etc. When the type information is known for each reference, resolution decisions are restricted within references of the same type. Otherwise, the types may need to be discovered as well as part of the entity resolution process.

Traditional entity resolution approaches pose entity resolution as a pairwise decision problem over references based on their attribute similarity. It can also be posed as a ▸ graph clustering problem, where references are clustered together based on their attribute similarities and each cluster is taken to represent one underlying entity. Entity resolution approaches differ in how the similarities between references are defined and computed and how the resolution decisions are made based on these similarities. Traditionally, each pairwise decision is made independently of the others. For example, the decision to resolve the two *Wang* references from papers 1 and 3 would be made independently of the decision to resolve the two *Chen* references from the same papers.

The first improvement is to account for the similarity of the coauthor names when such relationships are available. However, this still does not consider the "entities" of the related references. For the two "Wang" references in the earlier example, the two "C. Chen" coauthors match regardless of whether they refer to *Chen1* or *Chen2*. The correct evidence to use here is that the "Chens" are not co-referent. In such a setting, in order to resolve the "W. Wang" references, it is necessary to *resolve* the "C Chen" references as well and not just consider their name similarity. In the collective relational entity resolution approach, resolutions are not made independently, but instead one resolution decision affects other resolutions via hyper-edges.

Below, we discuss the different entity resolution approaches in greater detail.

Attribute-Based Entity Resolution

As discussed earlier, exact matching of attributes does not suffice for entity resolution. Several sophisticated similarity measures have been developed for textual strings (Cohen et al. 2003; Chaudhuri et al. 2003) that may be used for unsupervised entity resolution. Finally, a weighted combination of the similarities over the different attributes for each reference is used to compute the attribute similarity between two references. An alternative is to use adaptive supervised algorithms that learn string ▸ similarity metrics from labeled data (Bilenko and Mooney 2003). In the traditional entity resolution approach (Fellegi and Sunter 1969; Cohen et al. 2003), similarity is

computed for each pair of references r_i, r_j based on their attributes, and only those pairs that have similarity above some threshold are considered co-referent.

Efficiency

Even the attribute-only approach to entity resolution is known to be a hard problem computationally, since it is infeasible to compare all pairs of references using expensive similarity measures. Therefore, efficiency issues have long been a focus for data cleaning, the goal being the development of inexpensive algorithms for finding approximate solutions. The key mechanisms for doing this involve computing the matches efficiently and employing techniques commonly called "blocking" to quickly find potential duplicates (Hernández and Stolfo 1995; Monge and Elkan 1997), using cheap and index-based similarity computations to rule out non-duplicate pairs. Sampling approaches can quickly compute cosine similarity between tuples for fast text-joins within an SQL framework (Gravano et al. 2003). Error-tolerant indexes can also be used in data warehousing applications to efficiently look up a small but "probabilistically safe" set of reference tuples as candidates for matching for an incoming tuple (Chaudhuri et al. 2003). Generic entity resolution frameworks also exist for resolving and merging duplicates as a database operator and minimize the number of record-level and feature-level operations (Menestrina et al. 2006).

Probabilistic Models for Pairwise Resolution

The groundwork for posing entity resolution as a probabilistic ▸ classification problem was done by Fellegi and Sunter (1969), who studied the problem of labeling pairs of records from two different files to be merged as "match" (M) or "non-match" (U) on the basis of agreement γ among their different fields or attributes. Given an agreement pattern γ, the conditional probabilities $P(\gamma|M)$ and $P(\gamma|U)$ of γ given matches and non-matches are computed and compared to decide whether the two references are duplicates or not. Fellegi and Sunter showed that the probabilities $P(\gamma|M)$ and $P(\gamma|U)$ of field agreements can be estimated without requiring labeled training data if the different field agreements are assumed to be independent. Winkler (2002) used the EM algorithm to estimate the probabilities without making the independence assumption.

Probabilistic Models for Relational Entity Resolution

Probabilistic models that take into account interaction between different entity resolution decisions through hyper-edges have been proposed for named-entity recognition in natural language processing and for citation matching (McCallum and Wellner 2004; Singla and Domingos 2004). Such ▸ relational learning approaches introduce a decision variable y_{ij} for every pair of references r_i and r_j, but instead of inferring the y_{ij}'s independently, use conditional random fields for joint reasoning. For example, the decision variables for the "Wang" references and the "Chen" references in papers 1 and 3 would be connected to each other; features and functions would be defined to ensure that they are more likely to take up identical values.

Such relational models are supervised and require labeled data to train the parameters. One of the difficulties in using a supervised method for resolution is constructing a good training set that includes a representative collection of positive and negative examples. Accordingly, unsupervised relational models have also been developed (Li et al. 2005; Pasula et al. 2003; Bhattacharya and Getoor 2006). Instead of introducing pairwise decision variables, this category of approaches uses generative models for references using latent entity labels. Note that, here, the number of entities is unknown and needs to be discovered automatically from the available references. Relationships between the references, such as co-mentions or co-occurrences, are captured using joint distributions over the entity labels.

All of these probabilistic models have been shown to perform well in practice and have the

advantage that the match/non-match decisions do not depend on any user-specified similarity measures and thresholds but are learned directly from data. However, this benefit comes at a price. Inference in relational probabilistic models is an expensive process. Exact inference is mostly intractable and approximate strategies such as loopy belief propagation and Monte Carlo sampling strategies are employed. Even these approximate strategies take several iterations to converge, and extending such approaches to large datasets is still an open problem.

Other Approaches for Relational Entity Resolution

Alternative approaches (Bhattacharya and Getoor 2007; Kalashnikov et al. 2005; Dong et al. 2005) consider relational structure of the entities for data integration but avoid the complexity of probabilistic inference. By avoiding a formal probabilistic model, these approaches can handle complex and longer-range relationships between different entity references, and the resolution process is significantly faster as well. Such approaches also create pairwise decision nodes between references and create a dependency graph over them to capture the relationships in the data. But instead of performing probabilistic inference, they keep updating the value associated with each decision node by propagating relational evidence from one decision node to another over the dependency graph.

When the relationships between the references and the entities can be captured in a single graph, the matching entity for a specific reference may be identified using path-based similarities between their corresponding nodes in the graph. The connection strength associated with each edge in the graph can be determined in the unsupervised fashion given all the references, their candidate entity choices, and the relationships between them, by solving a set of nonlinear equations (Kalashnikov et al. 2005). This approach is useful for incremental data cleaning when the set of entities currently in the database is known and an incoming reference needs to be matched with one of these entities.

An alternative approach to performing collective entity resolution using relational evidence is to perform collective relational clustering (Bhattacharya and Getoor 2007). The goal here is to cluster the references into entities by taking into account the relationships between the references. This is achieved by defining a similarity measure between two clusters of references that take into account not only the attribute similarity of the references in the two clusters but also the neighboring clusters of each cluster. The neighboring clusters of any reference cluster c are defined by considering the references r' connected to references r belonging to c via hyper-edges and the clusters to which these related references belong. If the $r.C$ represents the current cluster for reference c, then $N(c) = \bigcup r'.C$, where $r.H = r'.H$ and $r.C = c$. For instance, the neighboring clusters for a $Wang$ cluster in our example containing the $Wang$ references from papers 1, 2, and 4 are the $Ansari$ cluster and the $Chen$ clusters containing the other references from the same papers. The relational similarity between two clusters is then computed by comparing their neighborhoods. This relational similarity complements attribute similarity in the combined similarity between two clusters. Intuitively, two entities are likely to be the same if they are similar in attributes and are additionally connected to the same other entities. Collective relational clustering can be efficiently implemented by maintaining a priority queue for merge-able cluster pairs and updating the "neighboring" queue elements with every merge operation.

Applications

Data cleaning and reference disambiguation approaches have been applied and evaluated in a number of domains. The earliest applications were on medical data. Census data is an area where detection of duplicates poses a significant challenge and Winkler (2002) has successfully applied his research and other baselines to this domain. A great deal of work has been done

making use of bibliographic data (Pasula et al. 2003; Singla and Domingos 2004; Bhattacharya and Getoor 2007). Almost without exception, the focus has been on the matching of citations. Work in co-reference resolution and disambiguating entity mentions in natural language processing (McCallum and Wellner 2004) has been applied to text corpora and newswire articles like the TREC corpus. There have also been significant applications in information integration in data warehouses (Chaudhuri et al. 2003).

Cross-References

- ▸ Classification
- ▸ Data Preparation
- ▸ Graph Clustering
- ▸ Record Linkage
- ▸ Similarity Measures
- ▸ Statistical Relational Learning

Recommended Reading

Bhattacharya I, Getoor L (2006) A latent dirichlet model for unsupervised entity resolution. In: The SIAM international conference on data mining (SIAM-SDM), Bethesda

Bhattacharya I, Getoor L (2007) Collective entity resolution in relational data. ACM Trans Knowl Discov Data 1(1):5

Bilenko M, Mooney RJ (2003) Adaptive duplicate detection using learnable string similarity measures. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining (KDD-2003), Washington, DC

Chaudhuri S, Ganjam K, Ganti V, Motwani R (2003) Robust and efficient fuzzy match for online data cleaning. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data, San Diego, pp 313–324

Cohen WW, Ravikumar P, Fienberg SE (2003) A comparison of string distance metrics for name-matching tasks. In: Proceedings of the IJCAI-2003 workshop on information integration on the web, Acapulco, pp 73–78

Dong X, Halevy A, Madhavan J (2005) Reference reconciliation in complex information spaces. In: The ACM international conference on management of data (SIGMOD), Baltimore

Fellegi IP, Sunter AB (1969) A theory for record linkage. J Am Stat Assoc 64:1183–1210

Gravano L, Ipeirotis P, Koudas N, Srivastava D (2003) Text joins for data cleansing and integration in an rdbms. In: 19th IEEE international conference on data engineering, Bangalore

Hernández MA, Stolfo SJ (1995) The merge/purge problem for large databases. In: Proceedings of the 1995 ACM SIGMOD international conference on management of data (SIGMOD-95), San Jose, pp 127–138

Kalashnikov DV, Mehrotra S, Chen Z (2005) Exploiting relationships for domain-independent data cleaning. In: SIAM international conference on data mining (SIAM SDM), Newport Beach, 21–23 Apr 2005

Li X, Morie P, Roth D (2005) Semantic integration in text: from ambiguous names to identifiable entities. AI Mag Spec Issue Semant Integr 26(1):45–58

McCallum A, Wellner B (2004) Conditional models of identity uncertainty with application to noun coreference. In: NIPS, Vancouver

Menestrina D, Benjelloun O, Garcia-Molina H (2006) Generic entity resolution with data confidences. In: First Int'l VLDB workshop on clean databases, Seoul

Monge AE, Elkan CP (1997) An efficient domain-independent algorithm for detecting approximately duplicate database records. In: Proceedings of the SIGMOD 1997 workshop on research issues on data mining and knowledge discovery, Tuscon, pp 23–29

Pasula H, Marthi B, Milch B, Russell S, Shpitser I (2003) Identity uncertainty and citation matching. In: Advances in neural information processing systems 15, Vancouver. MIT, Cambridge

Singla P, Domingos P (2004) Multi-relational record linkage. In: Proceedings of 3rd workshop on multi-relational data mining at ACM SI GKDD, Seattle

Winkler WE (2002) Methods for record linkage and Bayesian networks. Technical report, Statistical Research Division, U.S. Census Bureau, Washington, DC

EP

- ▸ Expectation Propagation

Epsilon Cover

Thomas Zeugmann
Hokkaido University, Sapparo, Japan

Motivation and Background

Epsilon covers were introduced in calculus. So we provide here a very general definition.

Definition

Let (M, ϱ) be a metric space, let $S \subseteq M$, and let $\varepsilon > 0$. A set $E \subseteq M$ is an *ε-cover* for S, if for every $s \in S$ there is an $e \in E$ such that $\varrho(s, e) \leq \varepsilon$.

An *ε-cover* E is said to be *proper*, if $E \subseteq S$.

Application

The notion of an ε-cover is frequently used in kernel-based learning methods.

For further information, we refer the reader to Herbrich (2002).

Cross-References

- ▸ Statistical Machine Translation
- ▸ Support Vector Machines

Recommended Reading

Herbrich R (2002) Learning kernel classifiers: theory and algorithms. MIT, Cambridge

Epsilon Nets

Thomas Zeugmann
Hokkaido University, Sapparo, Japan

Motivation and Background

Epsilon nets were introduced by Haussler and Welz (1987), and their usefulness for computational learning theory has been discovered by Blumer et al. (1989).

Let $X \neq \emptyset$ be any learning domain and let $\mathcal{C} \subseteq \wp(X)$ be any nonempty concept class. For the sake of simplicity, we also use $\mathcal{C}$ here as hypothesis space. In order to guarantee that all probabilities considered below do exist, we restrict ourselves to *well-behaved* concept classes (see ▸ PAC Learning).

Furthermore, let D be any arbitrarily fixed probability distribution over the learning domain X and let $c \in \mathcal{C}$ be any fixed concept.

A hypothesis $h \in \mathcal{C}$ is said to be *bad* for c iff

$$d(c, h) = \sum_{x \in c \triangle h} D(x) > \varepsilon.$$

Furthermore, we use

$$\Delta(c) =_{df} \{h \triangle c \mid h \in \mathcal{C}\}$$

to denote the set of all possible *error regions* of c with respect to $\mathcal{C}$ and D. Moreover, let

$$\Delta_\varepsilon(c) =_{df} \{h \triangle c \mid h \in \mathcal{C},\ d(c, h) > \varepsilon\}$$

denote the set of all *bad error regions* of c with respect to $\mathcal{C}$ and D.

Now we are ready to formally define the notion of an ε-net.

Definition

Let $\varepsilon \in (0, 1)$, and let $S \subseteq X$. The set S is said to be an *ε-net for* $\Delta(c)$ iff $S \cap r \neq \emptyset$ for all $r \in \Delta_\varepsilon(c)$.

Remarks

Conceptually, a set S constitutes an ε-net for $\Delta(c)$ iff every bad error region is hit by at least one point in S.

Example

Consider the one-dimensional Euclidean space $\mathbb{E}$, and let $X = [0, 1] \subseteq \mathbb{E}$. Furthermore, let $\mathcal{C}$ be the set of all closed intervals $[a, b] \subseteq [0, 1]$. Consider any fixed $c \in \mathcal{C}$, and let D be the uniform distribution, i.e., $D([a, b]) = 1/(b - a)$ for all $[a, b] \in \mathcal{C}$. Furthermore, let $h \in \mathcal{C}$; then we

may write $c \triangle h = I_1 \cup I_2$, where $I_1, I_2 \in \mathcal{C}$. Let $\varepsilon \in (0, 1)$ be arbitrarily fixed, and let

$$S = \{k\varepsilon/2 \mid 0 \leq k \leq \lceil 2/\varepsilon \rceil,\ k \in \mathbb{N}\}\ .$$

Then, S forms an ε-net for $\Delta(c)$. This can be seen as follows. Assume $r \in \Delta_\varepsilon(c)$. Then, $D(I_1) > \varepsilon/2$ or $D(I_2) > \varepsilon/2$. Now, by the definition of S, it is obvious that $D(I_i) > \varepsilon/2$ implies $I_i \cap S \neq \emptyset$, $i = 1, 2$.

Application

Recall that in ▸ PAC Learning, the general strategy to design a learner has been to draw a sufficiently large finite sample and then to find a hypothesis that is consistent with it. For showing that this strategy is always successful, the notion of an ε-net plays an important role. This can be expressed by the following observation.

Observation. Let $S = \{x_1, \ldots, x_m\}$ be an ε-net for $\Delta(c)$, and let $h \in \mathcal{C}$ be any hypothesis such that $h(x_i) = c(x_i)$ for all $1 \leq i \leq m$, i.e., h is consistent. Then we have $d(c, h) \leq \varepsilon$.

It then remains to show that the ▸ VC Dimension of $\mathcal{C}$ and of $\Delta(c)$ are the same and to apply Sauer's lemma to complete the proof.

For further information, we refer the reader to Blumer et al. (1989) as well as to Kearns and Vazirani (1994).

Cross-References

▸ PAC Learning
▸ VC Dimension

Recommended Reading

Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM 36(4):929–965

Haussler D, Welz E (1987) Epsilon nets and simplex range queries. Discret & Comput Geom 2:127–151 (1987)

Kearns MJ, Vazirani UV (1994) An introduction to computational learning theory. MIT, Cambridge

Equation Discovery

Ljupčo Todorovski
University of Ljubljana, Ljubljana, Slovenia

Synonyms

Computational discovery of quantitative laws; Symbolic regression

Definition

Equation discovery is a machine learning task that deals with the problem of learning quantitative laws and models, expressed in the form of equations, in collections of measured numeric data. Equation discovery methods take at input a ▸ data set consisting of measured values of a set of numeric variables of an observed system or phenomenon. At output, equation discovery methods provide a set of equations, such that, when used to calculate the values of system variables, the calculated values closely match the measured ones.

Motivation and Background

Equation discovery methods can be used to solve complex modeling tasks, i.e., establishing a mathematical model of an observed system. Modeling tasks are omnipresent in many scientific and engineering domains.

Equation discovery is strongly related to *system identification*, another approach to mathematical modeling. System identification methods work under the assumption that the structure of the model (the form of the model equations) is known or comes from a well-defined class of model structures, such as polynomials or neural networks. Therefore, they are mainly concerned with the parameter estimation task, that is, the task of determining the values of the model parameters that minimize the discrepancy between measured data and data obtained by simulating

the model. Equation discovery methods, on the other hand, aim at identifying both, an adequate structure of the model equations and appropriate values of the model parameters.

▸ Regression also deals with building predictive models from numeric data. The focus of regression methods is on building descriptive black-box models that can reconstruct the training data with high accuracy. In contrast, equation discovery methods focus on establishing explanatory models that, beside accurate predictions, provide explanations of the mechanisms that govern the behavior of the modeled system.

Early equation discovery methods dealt with rediscovering empirical laws from the history of science (this is where the synonym "computational discovery of quantitative laws" comes from). Through the years, the focus of the equation discovery methods has shifted from discovering quantitative laws to modeling real-world systems.

Structure of the Learning System

The task of equation discovery can be decomposed into two closely coupled subtasks of structural identification and parameter estimation. The first task of structural identification deals with the problem of finding the optimal structure of an equation. The second task of parameter estimation deals with the problem of finding the optimal values of the constant parameters in the equation. General approaches to and specific methods for equation discovery use different techniques to solve these two subtasks.

Approaches and Methods

There are two general and fundamentally different approaches to equation discovery. The first approach relies on a definition of a space of candidate equation structures. Following this definition, a generate-and-test (or ▸ learning as search) approach is used to generate different equation structures, solve the parameter estimation task for each of them, and report those equations that most closely approximate the data. The second approach relies on heuristics, used by scientists and engineers in the discovery or modeling processes, to establish an appropriate equation structure.

The first equation discovery system, Bacon (Langley 1981), follows the second approach described above. It incorporates a set of data-driven heuristics for detecting regularities (constancies and trends) in measured data and for formulating hypotheses based on them. An example heuristic would, when faced with a situation where the values of two observed variables increase/decrease simultaneously, introduce a new equation term by multiplying them. Furthermore, Bacon builds equation structure at different levels of description. At each level of description, all but two variables are held constant and hypotheses connecting the two changing variables are considered. Using a relatively small set of data-driven heuristics, Bacon is able to rediscover a number of physical laws including the ideal gas law, the law of gravitation, the law of refraction, and Black's specific heat law.

An alternative set of heuristics for equation discovery can be derived from dimensional analysis that is routinely used to check the plausibility of equations by using rules that specify the proper ways to combine variables and terms with different *measurements units*, different measurement scales, or types thereof. Following these rules, equation discovery method Coper (Kokar 1986) considers only equation structures that properly combine variables and constants, given the knowledge about their exact measurement units. Equation discovery method SDS (Takashi and Hiroshi 1998) extends Coper to cases, where the exact measurement units of the variables and constants involved in the equation are not known, but only knowledge about the types of the ▸ measurement scales is available.

Finally, the heuristics and design of the equation discovery method E* (Schaffer 1993) is based on a systematic survey of more than a hundred laws and models published in the Physical Review journal. The review shows that many of the published laws and models follow one of five different equation structures. By including only these five structures as its main heuristic for solving the structure identification task (implementing it as a ▸ language bias), E* was able to

reconstruct the correct laws and models in about a third of the test cases collected from the same journal.

Abacus (Falkenhainer and Michalski 1990) was the first equation discovery method that followed the generate-and-test (or ▸ learning as search) approach, mentioned above. Abacus experimented with different search strategies within a fixed space of candidate equation structures. Other methods that follow the generate-and-test approach differ in the ways they define the space of candidate equation structures and solve the parameter estimation task.

Equation discovery methods EF (Zembowitz and Zytkow 1992) and Lagrange (Džeroski and Todorovski 1995) explore the space of polynomial equation structures that are linear in the constant parameters, so they apply ▸ linear regression to estimate parameters. The user can shape the space of candidate structures by specifying parameters, such as, the maximal polynomial degree, the maximal number of multiplicative terms included in a polynomial, and a set of functions that can be used to transform the original variables before combining them into multiplicative terms.

While all of the above methods assume a fixed predefined ▸ language bias (via specification of the class of candidate equation structures or via heuristics for establishing appropriate structure), equation discovery method Lagramge (Todorovski and Džeroski 1997) employs dynamic declarative ▸ language bias, that is, let the user of the equation discovery method choose or specify the space of candidate equation structures. In its first version, Lagramge uses the formalism of context-free grammars for specifying the space of equation structures. The formalism has been shown to be general enough to allow users to build their specification upon many different types of modeling knowledge, from measurement units to very specific knowledge about building models in a particular domain of interest (Todorovski and Džeroski 2007). For solving the structure identification task, Lagramge defines a refinement operator that orders the search space of candidate equation structures, defined by the user-specified grammar, from the simplest ones to more complex. Exhaustive and ▸ beam search strategies are then being employed to the search space and for each structure considered during the search, Lagramge uses gradient-descent methods for nonlinear optimization to solve the parameter estimation task. The heuristic function that guides the search is based on the ▸ mean squared error that measures the discrepancy between the measured and simulated values of the observed system variables. Alternatively, Lagramge can use heuristic function that takes into account the complexity of the equation and is based on the ▸ minimum description length principle.

Successors of Lagramge, equation discovery methods, Lagramge 2 (Todorovski and Džeroski 2007), IPM (Bridewell et al. 2008), and HIPM (Todorovski et al. 2005), primarily focus on the improvement of the knowledge representation formalism used to formalize the modeling knowledge and transform it to ▸ language bias for equation discovery. All of them follow the paradigm of ▸ inductive process modeling.

Types of Equations

At first, equation discovery methods dealt with the problem of learning algebraic equations from data. Equation discovery method Lagrange (Džeroski and Todorovski 1995) extended the scope of equation discovery to modeling dynamics from ▸ time series data with ordinary differential equations. It took a naïve approach based on transforming the task of discovering ordinary differential equations to the simpler task of discovering algebraic equations, by extending the set of observed system variables with numerically calculated time derivatives thereof. By doing so, any of the existing equation discovery methods could be, in principle, used to discover differential equations. However, the naïve approach has a major drawback of introducing large numerical errors, due to instability of methods for numerical differentiation. Equation discovery method GoldHorn (Križman et al. 1995) replaced the instable numerical differentiation with the stable numerical methods for the inverse problem of

integration. Goldhorn also upgrades Lagrange with filtering methods to cope with measurement errors and noisy data.

While ordinary differential equations can model systems that change their state along a single dimension, time, partial differential equations can be used to model systems that change along many (temporal and spatial) dimensions. The naïve approach of introducing numerically calculated partial derivatives has been used in the Paddles (Todorovski et al. 2000) method for discovery of partial differential equations. The method first slices the measurement data into narrow spatial subsets, induces ordinary differential equations in each of them, and uses most frequently obtained equation structures to extend them with partial derivatives and to obtain a relatively small class of partial differential equation structures to explore. All the equation discovery tasks in Paddles are solved using Lagramge (Todorovski and Džeroski 1997).

Applications

Equation discovery methods have been applied to various tasks of discovering equation-based laws and models from measured and/or simulation data. Application domains range from physics (mechanical and electrical engineering, fluid dynamics) (Takashi and Hiroshi 1998; Todorovski and Džeroski 1997, 2007), through ecology (population dynamics) (Todorovski and Džeroski 2007; Todorovski et al. 2005) to biochemistry (chemical kinetics) (Džeroski and Todorovski 2008; Langley et al. 2006).

Cross-References

- ▸ Identification
- ▸ Inductive Process Modeling
- ▸ Language Bias
- ▸ Learning as Search
- ▸ Linear Regression
- ▸ Measurement Scales
- ▸ Regression

Recommended Reading

Bridewell W, Langley P, Todorovski L, Džeroski S (2008) Inductive process modeling. Mach Learn 71(1):1–32

Džeroski S, Todorovski L (1995) Discovering dynamics: from inductive logic programming to machine discovery. J Intell Inf Syst 4(1): 89–108

Džeroski S, Todorovski L (2008) Equation discovery for systems biology: finding the structure and dynamics of biological networks from time course data. Curr Opin Biotechnol 19:1–9

Falkenhainer B, Michalski R (1990) Integrating quantitative and qualitative discovery in the ABACUS system. In: Kodratoff Y, Michalski R (eds) Machine learning: an artificial intelligence approach. Morgan Kaufmann, San Mateo

Kokar MM (1986) Determining arguments of invariant functional descriptions. Mach Learn 1(4): 403–422

Križman V, Džeroski S, Kompare B (1995) Discovering dynamics from measured data. Electrotech Rev 62(3–4):191–198

Langley P (1981) Data-driven discovery of physical laws. Cogn Sci 5(1):31–54

Langley P, Shiran O, Shrager J, Todorovski L, Pohorille A (2006) Constructing explanatory process models from biological data and knowledge. Artif Intell Med 37(3):191–201

Schaffer C (1993) Bivariate scientific function finding in a sampled, real-data testbed. Mach Learn 12(1–3):167–183

Takashi W, Hiroshi M (1998) Discovery of first-principle equations based on scale-type-based and data-driven reasoning. Knowl-Based Syst 10(7):403–411

Todorovski L, Bridewell W, Shiran O, Langley P (2005) Inducing hierarchical process models in dynamic domains. In: Veloso MM, Kambhampati S (eds) Proceedings of the twentieth national conference on artificial intelligence, Pittsburgh

Todorovski L, Džeroski S (1997) Declarative bias in equation discovery. In: Fisher DH (ed) Proceedings of the fourteenth international conference on machine learning, Nashville

Todorovski L, Džeroski S (2007) Integrating domain knowledge in equation discovery. In Džeroski S, Todorovski L (eds) Computational discovery of scientific knowledge. LNCS, vol 4660. Springer, Berlin

Todorovski L, Džeroski S, Srinivasan A, Whiteley J, Gavaghan D (2000) Discovering the structure of partial differential equations from example behaviour. In: Langley P (ed) Proceedings of the seventeenth international conference on machine learning, Stanford

Zembowitz R, Zytkow J (1992) Discovery of equations: experimental evaluation of convergence. In: Swartout WR (ed) Proceedings of the tenth national conference on artificial intelligence, San Jose

Error

▸ Error Rate

Error Correcting Output Codes

Synonyms

▸ ECOC

Definition

Error correcting output codes are an ▸ ensemble learning technique. It is applied to a problem with multiple classes, decomposing it into several binary problems. Each class is first encoded as a binary string of length T, assuming we have T models in the ensemble. Each model then tries to separate a subset of the original classes from all the others. For example, one model might learn to distinguish "class A" from "not class A." After the predictions, with T models we have a binary string of length T. The class encoding that is closest to this binary string (using Hamming distance) is the final decision of the ensemble.

Recommended Reading

Kong EB, Dietterich TG (1995) Error-correcting output coding corrects bias and variance. In: International conference on machine learning, Tahoe City

Error Curve

▸ Learning Curves in Machine Learning

Error Rate

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

Synonyms

Error

Definition

Error rate refers to a measure of the degree of prediction error of a model made with respect to the true model.

The term *error rate* is often applied in the context of ▸ classification models. In this context, *error rate* $= \mathrm{P}(\lambda(X) \neq Y)$, where XY is a joint distribution and the classification model λ is a function $X \rightarrow Y$. Sometimes this quantity is expressed as a percentage rather than a value between 0.0 and 1.0.

The error rate of a model is often assessed or estimated by applying it to ▸ test data for which the class labels (Y values) are known. The error rate of a classifier on test data may be calculated as *number of incorrectly classified objects/total number of objects*. Alternatively, a smoothing function may be applied, such as a ▸ Laplace estimate or an m-estimate.

Error rate is directly related to ▸ accuracy, such that *error rate* $= 1.0 -$ *accuracy* (or when expressed as a percentage, *error rate* $= 100 -$ *accuracy*).

Two common measures of *error rate* for ▸ regression models are ▸ mean squared error and ▸ mean absolute error.

Cross-References

▸ Accuracy
▸ Confusion Matrix
▸ Mean Absolute Error
▸ Mean Squared Error

Error Squared

Synonyms

▸ Squared error

Definition

Error squared is a common ▸ loss function used with ▸ regression. This is the square of the difference between the predicted and true values.

Error-Correcting Output Codes (ECOC)

▸ Class Binarization

Estimation of Density Level Sets

▸ Density-Based Clustering

Evaluation

Evaluation is a process that assesses some property of an artifact. In machine learning, two types of\break artifacts are most commonly evaluated, models and {algorithms}. ▸ Model evaluation often focuses on the predictive efficacy of the model, but may also assess factors such as its complexity, the ease with which it can be understood, or the computational requirements for its application. ▸ Algorithm evaluation often focuses on evaluation of the models an algorithm produces, but may also appraise its computational efficiency.

Evaluation Data

▸ Test Data
▸ Test Set

Evaluation of Learning Algorithms

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Victoria, Australia

Abstract

It is often desirable to assess the properties of a learning algorithm. Frequently such evaluation take the form of comparing the relative suitability of a set of algorithms for a specific task or class of tasks. Learning algorithm evaluation is the process of performing such assessment of a learning algorithm.

Synonyms

Algorithm Evaluation; Learning Algorithm Evaluation

Definition

Learning algorithm evaluation is the process of assessing a property or properties of a learning algorithm.

Motivation and Background

It is often valuable to assess the efficacy of a learning algorithm. In many cases, such assessment is relative, that is, evaluating which of several alternative algorithms is best suited to a specific application.

Processes and Techniques

Many learning algorithms have been proposed. In order to understand the relative merits of these alternatives, it is necessary to evaluate them. The primary approaches to evaluation can be characterized as either theoretical or experimental. Theoretical evaluation uses formal methods to

infer properties of the algorithm, such as its computational complexity (Papadimitriou 1994), and also employs the tools of computational learning theory to assess learning theoretic properties. Experimental evaluation applies the algorithm to learning tasks in order to study its performance in practice.

There are many different types of property that may be relevant to assess depending upon the intended application. These include algorithmic properties, such a time and space complexity. These algorithmic properties are often assessed separately with respect to performance when learning a model, that is, at *training time*, and performance when applying a learned model, that is, at *test time*.

Other types of property that are often studied are the properties of the models that are learned (see ▸ model evaluation). Strictly speaking, such properties should be assessed with respect to a specific application or class of applications. However, much machine learning research includes experimental studies in which algorithms are compared using a set of data sets with little or no consideration given to what class of applications those data sets might represent. It is dangerous to draw general conclusions about relative performance on any application from relative performance on such a sample of some unknown class of applications. Such experimental evaluation has become known disparagingly as a *bake-off*.

An approach to experimental evaluation that may be less subject to the limitations of bake-offs is the use of experimental evaluation to assess a learning algorithm's bias and variance profile. Bias and variance measure properties of an algorithm's propensities in learning models rather than being directly properties of the models that are learned. Hence they may provide more general insights into the relative characteristics of alternative algorithms than do assessments of the performance of learned models on a finite number of applications. One example of such use of bias-variance analysis is found in Webb (2000).

Techniques for experimental algorithm evaluation include bootstrap sampling, cross validation, and holdout evaluation.

Cross-References

▸ Model Evaluation

Recommended Reading

Hastie T, Tibshirani R, Friedman J (2001) The elements of statistical learning. Springer, New York

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

Papadimitriou CH (1994) Computational complexity. Addison-Wesley, Reading

Webb GI (2000) MultiBoosting: a technique for combining boosting and wagging. Mach Learn 40(2):159–196

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, Amsterdam/Boston

Evaluation of Model Performance

▸ Model Evaluation

Evaluation Set

▸ Test Set

Event Extraction from Media Texts

Gregor Leban[1], Blaž Fortuna[1], and Marko Grobelnik[2]
[1]Jozef Stefan Institute, Ljubljana, Slovenia
[2]Artificial Intelligence Laboratory, Jožef Stefan Insitute, Ljubljana, Slovenia

Abstract

The chapter describes the topic of using news content to automatically detect world events mentioned in the news. Various tasks required for identifying events are presented, such as semantic annotation, article clustering and

cross-lingual cluster matching. Given the identified events we also describe how date, location, relevant entities and other core event details can be determined automatically.

Definition

Event extraction from text is an area of research that involves identifying mentions of significant world events described in media documents, such as news articles. The goal is to identify the world events and extract as much information as possible about them in a structured form. Ideally, the extracted information should contain details about what happened, when, where, and who was involved in the event. Since relevant events are reported in numerous articles, the methods for detection of events can exploit this fact when identifying events and extracting event properties.

Motivation and Background

News outlets produce large amounts of news content every day. Most of the news articles describe recent happenings in the world, such as meetings of important politicians, natural disasters, societal issues, sports events, or pastimes of celebrities. The importance of the generated news content varies significantly – news about an approaching hurricane can be considered as much more important and relevant than a news article about a party held by a local politician. For the purposes of this paper, we will call important happenings *events*. There is no objective way to distinguish between important and non-important news stories, but a practical approach that can be used is to treat news as important if it is being reported by several news publishers. For practical purposes we can therefore define an event as a happening that is being covered in news articles by several news publishers.

News articles are written in a natural language which makes them easy to understand by humans, but hard to process by computers. Understanding information being described in an article requires the use of common sense, common knowledge, implicit information, and knowledge on how to disambiguate. Since it's hard to extract knowledge from the articles, it is also difficult to perform accurate information retrieval. Imagine, for example, that you would like to learn from the news about the events that happened in Washington state in the last month. The word "Washington" is for the computer just a sequence of letters. One can use it to perform a keyword search, which will however return various articles – from the ones about the state of Washington, about any of the 40 cities names Washington, as well as numerous people who are also named Washington. Even if all articles would be relevant, they are not grouped – there would be tens or hundreds of articles describing the same event, and it would be up to the reader to find if an article describes some event you have already seen or not. Additionally, there would also be no summary of what the event was about – the reader would have to read the articles and learn about that himself.

To make learning about the events a more pleasant experience, we would like to convert the unstructured information expressed in news articles into a structured form that can be stored in a machine-readable way. This is not a trivial task and requires several steps of processing. These steps include syntactic analysis, semantic enrichment, entity linking, document clustering, and information extraction. The final result of the processing is a structured database of world events. Due to the extensive metadata, it is possible to find events based on the date, location, or relevant entities. Articles about an event are grouped together which helps significantly to reduce the information overload. A summary of an event can also be obtained by aggregating common information from multiple news articles.

To our knowledge, there are at least three systems that are identifying world events by analyzing news media. GDELT project (Gao et al. 2013; Leetaru et al. 2013) performs event detection by extracting information from individual sentences in the news articles. Since several events are potentially extracted from a single article, it contains a huge collection of events (over 200 million) that were extracted from 1979 to the

present. European media monitor (Steinberger et al. 2005; Pouliquen et al. 2008) focuses on the identification of current political events by combining and processing news articles in multiple languages. Event Registry (Leban et al. 2014a,b) similarly extracts world events with the additional metadata from articles in several languages. In the next sections, we will describe some of the core components that are needed by these systems.

Structure of Learning System

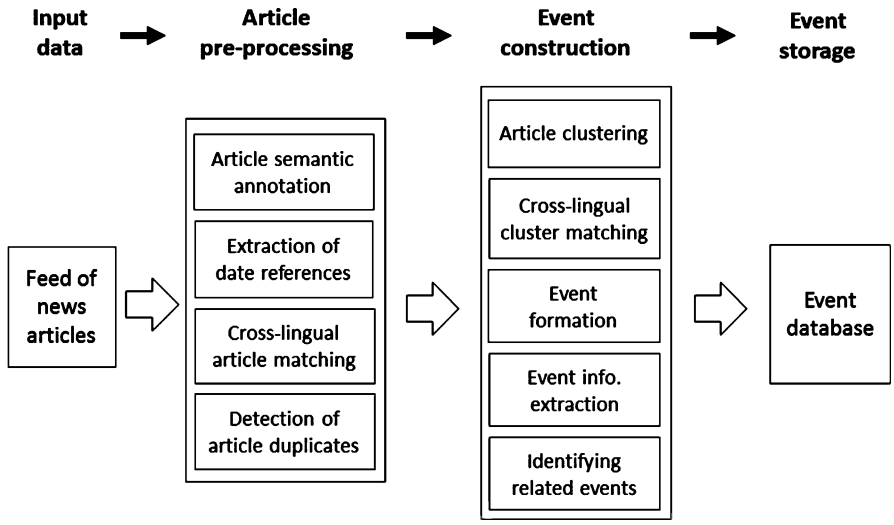
The identification of events from news articles requires a pipeline of services or components that provide specific functionalities. These services are shown in the Fig. 1 and will be described next.

News Collection

The first step in identification of events from news is to obtain the news content from a large set of news publishers. The content can be collected by either crawling the website of the news publishers or by identifying their RSS feeds and extracting article information from them. The use of RSS feeds, which are almost always available, is a better approach since they are significantly less data and time intensive compared to repeatedly crawling the whole websites. The RSS feeds do however often contain only article excerpts and crawling of the article page is therefore still needed. The main technical challenge with crawling the page is the identification of the article content and removal of the rest of the page. It is also important to extract as much metadata about the article as possible. This metadata can include the title of the article, publisher's name, date of publishing, author, etc.

Text Annotation

The collected news articles contain just plain text – there is no semantic information available about its content. In order to be able to extract semantic information about the described event, the text first needs to be annotated with semantic information that can be detected in the text. Common types of annotations are the named entities (people, locations, organizations) mentioned in the text and article topics. The challenge in annotation is twofold: first, the token or phrase that represents a named entity (such as "Paris") has to be identified, and second, it needs to be linked to a resource identifier that semantically represents the entity (such as a URI in a knowledge base). The first task is called



Event Extraction from Media Texts, Fig. 1 Various components required in the process of extracting events from news articles

named entity recognition and can be best solved using conditional random fields or more recently with convolutional neural networks. The second task is called entity linking and requires the use of a knowledge base containing an extensive set of relevant entities. Systems for entity linking such as Wikipedia Miner (Milne and Witten 2013) rely on the use of open knowledge bases such as DBpedia, Freebase, or YAGO.

An important type of data annotations are also temporal expressions mentioned in the text. Their detection is crucial in order to determine the date of the event that is being described in the news. Dates can be expressed in an absolute ("July 15th, 2015", "2015-07-12") or relative form ("yesterday morning," "last week"). The detection of absolute temporal expressions can be efficiently performed using a set of regular expressions. The relative expressions can either be identified using rule-based or sequence labeling approaches. The detected relative expressions can then be normalized into the absolute form using the article's publish date.

Clustering Approach to Event Identification

In order to identify events, a clustering approach can then be applied in order to group similar articles that describe the same event. The reasoning behind using clustering is the reasonable assumption that if articles are describing the same event, they will share similar vocabulary and mention similar entities. The most valuable features for the clustering algorithm are therefore the article text itself as well as the detected named entities and topics. The article text can be transformed into the bag-of-words form where each term in the document is normalized according to a chosen weighting scheme, such as TF-IDF. A feature vector can be generated for each article by concatenating the weights of the article terms and the mentioned named entities. A similarity measure, such as cosine similarity, can then be used to compute the similarity between individual articles.

Given the article feature vectors and the similarity measure, the clusters representing events can be identified using various clustering methods. European media monitor, for example, uses an agglomerative bottom-up clustering algorithm to group all articles published in a 24-h time window. Articles are grouped into the same cluster as long as their similarity is above the selected threshold. Centroid vectors of the obtained clusters are also compared with clusters identified on a previous day. The clusters that are found to be similar enough are merged and therefore represented as a single event. This allows the method to identify events that span across several days.

Event registry, on the other hand, uses an online approach to clustering. Each new article is clustered immediately after being added to the system. The clustering approach works as follows. The feature vector of the article is first being compared to the feature vectors of the centroids of all existing clusters. If the cosine similarity to the most similar centroid is above the selected threshold, the article is simply assigned to the cluster. Otherwise, a new, so-called *micro-cluster* is created containing only the single article. As new articles are added to the system, the micro-clusters can grow in size as articles are being added to them. Once they reach a certain size (depending on the language, this can be between three and ten articles), they are no longer considered as micro-clusters but instead as proper events. Micro-clusters that never reach the necessary size are not considered as events and are eventually removed. There are also different validation methods that are being called regularly in order to ensure the highest quality of the clusters. As clusters grow, for example, it can occur that the centroids of two clusters become more and more similar. One of the validation methods therefore checks different pairs of clusters and merges them in case their similarity, as measured by the cosine similarity of their centroid vectors, is above the threshold. Additionally, a separate method also checks each cluster if it is still sufficiently coherent or should instead be split into two separate clusters. The main idea behind splitting is to project all articles in the cluster onto a line and divide them into two groups depending on whether their projection was left or right of the centroid. This is repeated several times. In

the first iteration the first principal component of the original cluster is used as the projection line. In the following steps, the articles are projected onto the line that passes the centroids of the two groups obtained in the previous iteration. Once the two groups become stable, the method compares the original cluster and the two identified groups using the Bayesian information criterion in order to determine whether the cluster should be split or not. The last maintenance method is responsible for removing obsolete clusters. An event is reported in the media only for a limited number of days. To avoid assigning new articles to obsolete clusters, the method removes (micro-) clusters once the oldest member articles reach a certain age. In case of Event Registry, clusters are removed after they become 5 days old.

Both described approaches for identifying events using clustering have their advantages and disadvantages. The approach used in European media monitor works in batch mode which makes the identified events more stable. The downside is however that it is not suitable for real-time monitoring and detection of breaking events. On the other hand, the approach used by Event Registry can identify new events as soon as the sufficient number of articles about it has been written. However, because of the online mode of the algorithm, the identified clusters can be merged or split during their lifetime which makes them more volatile.

Cross Lingual Event Detection

Until this point we have not considered the fact that news articles are written in different languages. Since the described clustering approaches rely on the article text, the methods are evidently language dependent. It is not sensible, for example, to compute cosine similarity between an English and German article; therefore content from each language has to be clustered separately. As a result, events represented by the clusters will contain only articles in a single language. Since most events are reported in multiple languages, we want to find methods for identifying clusters in different languages that describe the same event. This will allow us to see how the same news is reported in different languages, what topics are more or less likely to break the language barrier, how fast does the information spreads through the languages, etc.

In order to link the appropriate clusters, we can represent the problem as a binary classification problem. Given a cluster pair c_1 and c_2 in languages l_1 and l_2, we need to compute a set of discriminative features that will help us to determine if both clusters describe the same event or not. A machine learning model can then be trained to classify the cluster pairs based on the values of the computed features.

One set of learning features can be computed by inspecting individual articles assigned to the clusters. Using a method such as canonical correlation analysis (CCA), it is possible to compute an estimated score of relatedness of two articles in different languages. The method is trained on a comparable corpus, which is a collection of documents in multiple languages, with alignment between documents that are on the same topic or even rough translations of each other. An example of such corpus is Wikipedia, where each entry can be described in multiple languages. Using the CCA, we can compare pairs of documents in the tested clusters c_1 and c_2 and compute features such as the maximum or the average score of similarity between the documents in the two clusters.

Additional set of important learning features can be computed by aggregating the annotated entities mentioned in the articles. Given the articles in each cluster, we can analyze how often do individual entities appear in the articles in order to estimate their relevance for the event – entities that appear more frequently can be considered as more important to the event compared to entities that are mentioned fewer times. One way to score an entity in a cluster is simply to compute the ratio of articles in the cluster that mention it. A more advance approach can also take into account the number of times the entity is mentioned in each article and its mentioned location – an entity is likely more relevant if it is mentioned at the beginning of the article than if at the end. Since entities are language independent (same entity, although mentioned in

different languages, is represented with the same identifier), we can construct for each cluster a weighted vector of relevant entities. For a pair of clusters, a similarity measure can again be used to compute similarity of the clusters according to the mentioned entities. Since events are mostly centered around entities, the similarity score can be an important feature when deciding if two clusters are about the same event or not.

Additionally, time similarity is also an important feature. If articles in one cluster were published in a similar time period as articles in another cluster, they are more likely about the same event as if they were published several days apart. If dates mentioned in the articles are being extracted, the ratio of common dates mentioned in the two clusters can also be a relevant feature.

In order to train the classification model, we first need the learning data. A human expert should therefore provide a set of positive and negative examples – cluster pairs that are about the same events as well as pairs that are not. For each cluster pair, values of the mentioned features can be computed and concatenated into a single feature vector. A machine learning classifier, such as SVM, can then be trained to best distinguish between the positive and negative examples based on the learning features. An experiment using the described approach (Rupnik et al. 2016) reports the cluster linking accuracy of 0.893 as measured using F1 score.

Extraction of Event Properties

Based on the described approach, an event consists of one or more clusters, where each cluster contains articles from a single language. As the final step, we wish to extract from the articles in the clusters as much structured information as possible about the event.

To determine the date of the event, we can analyze the publishing date of the articles in the clusters. The simplest method can be to use the date of the first article as the date of the event. This approach can generate erroneous results for events that are reported in advance (such as various meetings of politicians, product announcements, etc.) as well as when the collected publishing dates of the articles are potentially inaccurate. A more error-prone approach is to analyze the density of reporting and use the time point where the reporting intensified as the date of the event. Additional input can be provided by the mentioned date references – a particular date that is consistently mentioned across the articles is likely the correct date of the event.

In order to determine who is involved in the event, we can analyze and aggregate the entities mentioned in the articles. A list of relevant entities and their score of relevance can be obtained by analyzing the frequency of their occurrence in the articles as well as the locations of the mentions in text. Entities that appear in event's articles more frequently and early in the text are more important than entities that are just rarely mentioned and appear late. Entities can be scored and ranked according to this criterion which provides an accurate aggregated view on what and who is the event about.

Another core property of the event is also the location where the event occurred. Since the event location is commonly mentioned in the articles, we can identify it by analyzing the frequently mentioned entities that are of type location – knowledge about the entity type can be retrieved from the knowledge base used in entity linking. Additional signal for determining the location can be obtained by inspecting the datelines of the articles. A dateline is a brief piece of text at the beginning of the news article that describes where and when the described story happened. The problem with datelines is that they are not present in all news articles, and even when they are, they sometimes represent the location where the story was written and not the actual location of the event. To determine the event location, one can simply use the city that is mentioned the most in the articles. A more advanced approach can again rely on machine learning. Each city that is mentioned in the articles about an event can be considered as a candidate for the event location. For each city we therefore generate a set of features based on which a classification model can compute the probability that it is the location of the event. The features can be the number or ratio of times the city is mentioned in the articles, the number of times it is mentioned in

the dateline, how commonly the city is mentioned in all the articles, etc. To train the classification model, we again need the experts to manually provide information about the correct location of various events. Using the training data, we can then train a model that will classify each candidate city independently. Because they are evaluated independently, it is possible that the model finds several locations to be the event location. To choose the most likely city, it is important to use a probabilistic classifier that can also return a degree of certainty – in such cases, one can simply choose the location with the highest probability.

There are many other properties that could be extracted which are specific for individual event types. In case of an earthquake, for example, important properties would include the number of casualties and the strength of the earthquake. Similarly, for a football game, the relevant information would be the names of the teams that played and the final score. Identifying such properties and their values is a cumbersome task. It first requires that each event is classified into an event type (such as earthquake, football game, meeting, etc.). To perform classification, a taxonomy of event types is first needed together with a model that can perform classification into the taxonomy. Next, for each event type, a set of properties/slots need to be identified that are relevant for the event type. A pattern or rule-based approach can then be used to determine the values for these properties.

Cross-References

- ▸ Classification
- ▸ Clustering
- ▸ Cross-Lingual Text Mining
- ▸ Entity Resolution
- ▸ Text Mining

Recommended Reading

Gao J, Leetaru KH, Hu J, Cioffi-Revilla C, Schrodt P (2013) Massive media event data analysis to assess world-wide political conflict and instability. In: Social computing, behavioral-cultural modeling and prediction. Springer, Berlin/New York, pp 284–292

Leban G, Fortuna B, Brank J, Grobelnik M (2014a) Event registry: learning about world events from news. In: Proceedings of the companion publication of the 23rd international conference on World wide web companion, Seoul, pp 107–110

Leban G, Fortuna B, Brank J, Grobelnik M (2014b) Cross-lingual detection of world events from news articles. In: Proceedings of the 13th international semantic web conference, Trentino

Leetaru K, Schrodt PA (2013) GDELT: global data on events, location, and tone, 1979–2012. ISA Annu Conv 2:4

Milne D, Witten IH (2013) An open-source toolkit for mining Wikipedia. Artif Intell 194:222–239

Pouliquen B, Steinberger R, Deguernel O (2008) Story tracking: linking similar news over time and across languages. In: Proceedings of the workshop on multi-source multilingual information extraction and summarization, Manchester, pp 49–56

Rupnik J, Muhic A, Leban G, Škraba P, Fortuna B, Grobelnik M (2016) News Across Languages - Cross-Lingual Document Similarity and Event Tracking. J. Artif. Intell. Res., Special Track on Cross-language Algorithms and Applications 55, 283–316

Steinberger R, Pouliquen B, Ignat C (2005) Navigating multilingual news collections using automatically extracted information in: Proc. of the 27th International Conference on Information Technology Interfaces, pp. 25–32

Evolution of Agent Behaviors

▸ Evolutionary Robotics

Evolution of Robot Control

▸ Evolutionary Robotics

Evolutionary Algorithms

Synonyms

Evolutionary computation; Evolutionary computing; Genetic and evolutionary algorithms

Definition

Generic term subsuming all machine learning and optimization methods inspired by neo-Darwinian evolution theory.

Cross-References

- ▸ Coevolutionary Learning
- ▸ Compositional Coevolution
- ▸ Evolutionary Clustering
- ▸ Evolutionary Computation in Economics
- ▸ Evolutionary Computation in Finance
- ▸ Evolutionary Computational Techniques in Marketing
- ▸ Evolutionary Feature Selection and Construction
- ▸ Evolutionary Fuzzy Systems
- ▸ Evolutionary Games
- ▸ Evolutionary Kernel Learning
- ▸ Evolutionary Robotics
- ▸ Neuroevolution
- ▸ Nonstandard Criteria in Evolutionary Learning
- ▸ Test-Based Coevolution

Evolutionary Clustering

David Corne[1], Julia Handl[2], and Joshua Knowles[2]
[1]Herriot-Watt University, Edinburgh, UK
[2]University of Manchester, Manchester, UK

Synonyms

Cluster optimization; Evolutionary grouping; Genetic clustering; Genetic grouping

Definition

Evolutionary clustering refers to the application of evolutionary algorithms (also known as genetic algorithms) to data clustering (or cluster analysis), a general class of problems in machine learning with numerous applications throughout science and industry. Different definitions of data clustering exist, but it generally concerns the identification of homogeneous groups of data (clusters) within a given data set. That is, data items that are similar to each other should be grouped together in the same cluster or group, while (usually) dissimilar items should be placed in separate clusters. The output of any clustering method is therefore a specific collection of clusters. If we have an objective way to evaluate (calculate the quality of) a given grouping into clusters, then we can consider the clustering task as an optimization problem. In general, this optimization problem is NP hard, and it is common to address it with advanced heuristic or metaheuristic methods. Evolutionary algorithms are prominent among such methods and have led to a variety of promising and successful techniques for cluster optimization.

Motivation and Background

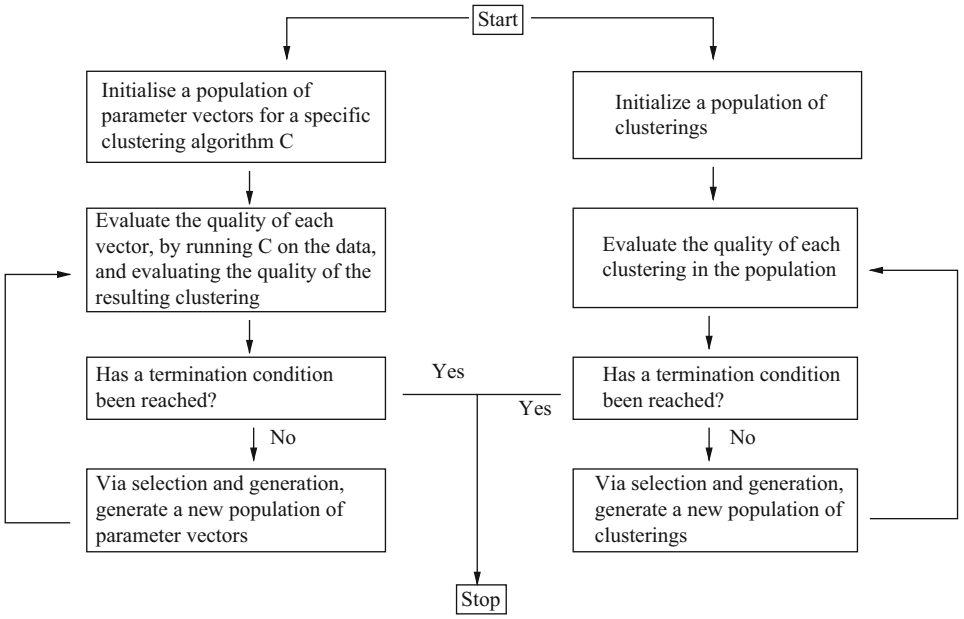
In many problem-solving scenarios, we have large amounts of data. We need to cluster those data sensibly into groups in order to help us understand the problem and decide how to proceed further (see clustering). It is common, in fact, for this initial "cluster analysis" stage to be the most important (or only) stage in the investigation. In bioinformatics, for example, a frequent activity is the clustering of gene expression data (data that indicate, for a specific cell, how active each of several thousands of genes are at different points in time or under different experimental conditions). A very important current challenge is to understand the role of each gene; by clustering such data, which means arranging genes into groups such that genes in the same group have similar patterns of activity, we find important clues about genes whose role is currently unknown, simply by assigning their putative role as being related to that of genes (whose role is known) that are in the same cluster. Meanwhile, a ubiquitous situation

in industry and commerce is the clustering of data about customers or clients. Here, the role of clustering is all about identifying what types of clients (e.g., based on age, income, postcode, and many other attributes that may make up a customer's profile) buy or use certain kinds of products and services. Effective ways to identify groups enable companies to better target their products and their direct marketing campaigns and/or make more effective decisions about loans, credit, and overdrafts. Many machine learning techniques can be used to predict things about customers, predict things about genes, and so forth. However, the value of clustering (in a similar way to visualization of the data) is that it can lead to a much deeper understanding of the data, which in turn informs the continuing process of applying machine learning methods to it. In this general context, there are many well-known and well-used clustering methods, such as k-means, hierarchical agglomerative clustering, neighbor joining, and so forth. However, there are also well-known difficulties with these methods; in particular, there is often a need to choose in advance the number of clusters to find in the data, and they tend to be strongly biased toward finding certain types of groupings. For these reasons, methods that are more flexible have been recently investigated, and evolutionary clustering techniques are prominent among these. They are flexible in that (e.g., unlike k-means) the choice of the number of clusters does not have to be made a priori, and the method is not tied to any particular way of identifying the distance between two items of data, nor is there any a priori inductive bias concerning what counts as a good clustering. That is, in broad terms, an evolutionary clustering algorithm allows a user to flexibly make these decisions in view of the actual problem at hand; these decisions are then "plugged into" the algorithm which proceeds to search for good clusterings.

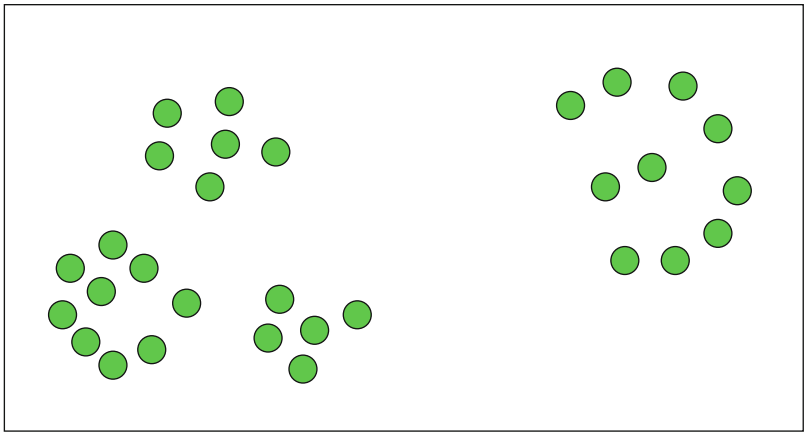
Given a data set to be clustered, the concept of evolutionary clustering covers two distinct ways in which we can address the problem of finding the best clustering. Each of these approaches is under continuing research and has proven successful under different conditions. The first approach is to use an evolutionary algorithm to search the space of candidate groupings of the data; this is the most straightforward approach and perhaps the most flexible in the sense discussed above. The second approach is to "wrap" an evolutionary algorithm around a simpler clustering algorithm (such as k-means) and either use the evolutionary algorithm to search the space of features for input to the clustering algorithm (i.e., the evolutionary algorithm is doing feature selection in this case) or to search a space of parameters, such as the number of clusters, feature weights, and/or other parameters of the clustering algorithm in use. Central in all of these approaches is a way to measure the quality of a clustering, which in turn depends on some given metric that provides a distance between any pair of data items. Although some applications come with pre-identified ways to measure distance and cluster quality, in the following we will assume the most common approach, in which distance is the Euclidean distance between the data items, and the measure of quality for a given clustering is some ratio of within-cluster and between-cluster similarities.

We illustrate the two main approaches to evolutionary clustering in Fig. 1.

There are several examples of the first type of approach, called "indirect" evolutionary clustering in Fig. 1 (left). This approach is often used where the "internal" clustering method ("C" in the figure) is very sensitive to initialization conditions and/or parameters of the metric in use to measure distance between items. For example, if C is the k-means algorithm, then, for each application of C, we need choices for the parameter k and for each of k initial cluster center positions in the data space. The parameter vectors referred to in the figure would be precisely these; the evolutionary algorithm searches this parameter space, finding those that lead to an optimized clustering from k-means. Figure 2 illustrates why this will often be a more effective approach than k-means alone. In this case, it is entirely unclear whether these data form two, four, or even five clusters. There are two widely separated groups of points, and this two-cluster solution may be easily found by a 2-means algorithm. However,



Evolutionary Clustering, Fig. 1 Evolutionary clustering. The two main approaches to evolutionary clustering: indirect (*left*) and direct (*right*)



Evolutionary Clustering, Fig. 2 An example data set with many potential interpretations of the number of clusters

to the human eye, there is also a clear four-cluster solution, further analysis of which may lead to better understanding of these data. This four-cluster solution is difficult for a 4-means algorithm to find, depending on very fortunate initial settings for the cluster centers. The embedding of k-means within an evolutionary algorithm allows for the iterative optimization of parameters and starting conditions to arrive at this optimal solution.

On the right in Fig. 1, we see the direct approach, in which the evolutionary algorithm searches the space of clusterings of the data. The key features in this approach are the encoding and genetic operators. After evaluating the quality of each of a population of clusterings, a new population is generated from the old one via selection and variation. Essentially, some individuals from the current population are treated as "parents," and new ones are produced from these by using genetic operators. The encoding dictates precisely how a specific data clustering is represented, while the operators determine how new clusterings are derived from

the old ones. To take a simple example, suppose we needed to cluster ten items (A, B, C,. . ., J) into an arbitrary number of groups. In a simple encoding, we might represent a clustering as a vector of ten labels, independently chosen from 1 to 10, in which the *i*th element gives the group label of the *i*th item. Hence, the following individual in our population of clusterings 2 3 5 5 1 5 7 3 2 7 would represent the following grouping: (A, I) (B, H) (C, D, F) (E) (G, I). Given such a representation, a typical genetic operator might be to randomly change a single label in a single parent. For example, we may choose the fifth element in the above vector and change it randomly to 7, effectively placing item E in the same group as items G and I. Further notes about operators for this and other encodings are given in a special subsection below. Going back to the example in Fig. 2, meanwhile, it is worth noting that there are potentially five clusters, as the group on the right can be perceived as a central group of two items, surrounded by a single backward-C-shaped group. The "backward-C" cluster is an example that cannot be reliably detected (as a distinct cluster from the group of two items contained within it) with most standard cluster analysis methods. Traditional approaches typically incorporate the assumption that clusters will be centered around a particular position, with the likelihood of a point belonging to that cluster falling monotonically with its distance from that position. One of the strengths of evolutionary clustering is that it provides the flexibility to work effectively with arbitrary definitions of what may constitute a valid cluster.

Objective Functions for Evolutionary Clustering

It can be strongly argued that the clustering problem is inherently multiobjective, yet most methods employ only a single performance criterion to optimize. In fact, there are at least three groups of criteria commonly used (but usually one at a time) in clustering (both evolutionary clustering and other methods). These are compactness, connectedness, and spatial separation. When an algorithm optimizes for compactness, the idea is that clusters should consist of highly homogeneous data items only – that is, the distance (or other measure of variation) between items in the same cluster should be small. In contrast, if we optimize the degree of connectedness, then we are increasing the extent to which neighboring data items should share the same cluster. This can deal with arbitrarily shaped clusters, but can lack robustness when there is little spatial separation between clusters. Finally, spatial separation is usually used as a criterion in combination with compactness or with a measure of the balance of cluster sizes.

In multiobjective clustering, the idea is to explicitly explore the solutions that are trade-offs between the conflicting criteria, exploiting the fact that these trade-off solutions are often the ones that most appeal as intuitively "correct" solutions to a clustering problem. Handl and Knowles (2007) introduced a multiobjective evolutionary algorithm, MOCK, which treats a clustering problem as a two-objective problem, using measures of compactness and connectedness for the two objectives. MOCK's multiobjective search process is based on the PESA-II evolutionary multiobjective optimizer (Corne et al. 2001). Following the use of MOCK for a clustering problem, an intermediate result (inherent in multiobjective optimization methods) is a (possibly large) collection of different clusterings. These will range from clusterings that score very well on compactness but poorly on connectedness through clusterings that achieve excellent connectedness at the expense of poor compactness. It is useful to note that the number of clusters tends to increase as we go from poor connectedness to high-connectedness clusters. Arguably, in many applications, such a collection of alternative solutions is useful for the decision-maker. Nevertheless, the MOCK approach incorporates an automated model selection process that attempts to choose an ideal clustering from the discovered approximate Pareto front. This process is oriented around the notion of determining the "right" number of clusters and makes use of Tibshirani et al. (2001) gap statistic

(full details are in Handl and Knowles 2007). Extensive comparison studies, using a wide variety of clustering problems and comparing with many alternative clustering methods, show consistent performance advantages for the MOCK's approach. Recent work has explored different objectives, encodings, and model selection mechanisms for multiobjective clustering, including the interpretation of the approximation set as a clustering ensemble (Handl and Knowles 2013).

Encodings and Operators for Evolutionary Clustering

The encoding methods used in indirect approaches to evolutionary clustering are fairly straightforward, as they only require the specification of the parameters (and, potentially, initialization points) for the clustering method(s) used. Arguably, the development of direct approaches to evolutionary clustering is more involved, as the choice of a suitable encoding method is nontrivial and has been shown to have significant impact on optimization performance.

Encodings range from the straightforward representation noted above (with the ith gene coding for the cluster membership of the ith data item) to more complex representations, such as matrix-based or permutation-based representations. Before providing a brief description of other encodings, it is worth briefly examining a well-known disadvantage of the simple encoding. Given that they have a population, evolutionary algorithms offer the opportunity to use multi-parent genetic operators – that is, we can design operators that produce a new candidate clustering given two or more "parent" clusterings. Such operators are neither mandatory nor necessarily beneficial in evolutionary algorithms, and there is much literature discussing their merits and how this depends on the problem at hand. However, they are often found helpful, especially in cases where we can see some intuitive merit in combining different aspects of parent solutions, resulting in a new solution that seems to have a chance at being good, but which we would have been immensely unlikely to obtain from single-parent operators given the current population. In this context, we can see, as follows, that the opposite seems to be the case when we use standard multi-parent operators with the simple encoding. Suppose the following are both very good clusterings of ten items:

Clustering 1: 1111122222
Clustering 2: 2222211111

Clearly, a good clustering of these items places items 1–5 together, and items 6–10 together, in separate groups. It is also clear, however, that using a standard crossover operator between these two parents (e.g., producing a child by randomly choosing between clusterings for each item in turn) will lead to a clustering that mixes items from these two groups, perhaps even combining them all into one group. The main point is that a crossover operation destroys the very relationships between the items that underpinned the fitness of the parents. One of the more prominent and influential representations for clustering, incorporating a design for far more effective multi-parent operators, was that of Falkenauer's "Grouping Genetic Algorithm," which also provides a general template for the implementation of evolutionary algorithms for grouping problems. The essential element of Falkenauer's method is that multi-parent operators recombine entire groups rather than item labels. For example, suppose we encode two clusterings explicitly as follows:

Clustering 3: (A,I,B,H)(C,G)(D,E,F,J)
Clustering 4: (A,I,B,H)(C,D,J)(E,F,G)

A Falkenauer-style crossover operator works as follows. First, we randomly choose some entire groups from the first parent and some entire groups from the second parent; the child in this case might then be:

(A,I,B,H)(C,G)(E,F,G)

in which the groups that come from the first parent are underlined. Typically, we will now have some repeated items; we remove the entire groups that contain these items and came from the first parent, in this case leaving us with:

(A,I,B,H)(E,F,G)

The final step is to add back the missing items, placing them one by one into one of the existing groups or perhaps forming one or more new groups. The application in hand will often suggest heuristics to use for this step. In clustering, for example, we could make use of the mean Euclidean distance from items in the groups so far. Whatever the end result in this case, note that the fact that A, I, B, and H were grouped together in both parents will be preserved in the child. Similarly, the E, F, G grouping is inherited directly from a parent.

A more recent and effective approach to encoding a clustering is one first proposed in Park and Song (1998) called a link-based encoding. In this approach, the encoding is simply a list of item indices and is interpreted as follows. If the ith element in the permutation is j, then items i and j are in the same group. So, for example,

B C E E A E G C B G

represents the following grouping:

(A,B,C,D,E,H,I)(F,G,J)

Standard crossover operators may be used with this encoding, causing (intuitively) a reasonable degree of exploration of the space of possible clusterings, yet preserving much of the essential "same-group" relationships between items that were present in the parents. In Handl and Knowles (2007) it is shown why this encoding is effective compared with some alternatives. We also briefly note other encodings that have been prominent in the history of this subfield. An early approach was that of Jones and Beltramo, who introduced a "permutation with separators" encoding. In this approach, a clustering is encoded by a permutation of the items to be clustered, with a number of separators indicating cluster boundaries. For example, if we have ten items to cluster (A–J) and use S as the separator, the following is a candidate clustering:

A I B H S C G S D E F J

representing the same grouping as that of "Clustering 3" above. Jones and Beltramo offered a variant of this encoding that is a cross between the direct and indirect approaches. In their greedy permutation encoding, a clustering is represented by a permutation (with no separator characters), with the following interpretation: the first k items in the permutation become the centers of the first k clusters. The remaining items, in the order they appear, are added to whichever cluster is best for that item according to the objective function (clustering quality metric) in use.

Applications for Evolutionary Clustering

Recent work on evolutionary clustering has focused on applications of evolu- tionary clustering to data-mining problems in a variety of disciplines, including market segmentation (by Ying, Sudha, Lusch, and Brusco) and social network analysis (by Pizutti). As mentioned above, evolutionary clustering brings key advantages in terms of its accuracy, but, possibly, its most important benefit lies in the flexibility of the approach. The capability to consider and explore trade-offs with respect to multiple clustering objectives opens up new opportu- nities for data integration, particularly in the context of exploratory analytics in applications that involve diverse, noisy (and sometimes poorly understood) data sources.

Cross-References

- ▸ Clustering
- ▸ Feature Selection
- ▸ Semi-supervised Learning
- ▸ Supervised Learning
- ▸ Unsupervised Learning

Recommended Reading

Cole RM (1998) Clustering with genetic algorithms. Masters dissertation, Department of Computer Science, University of Western Australia

Corne DW, Jerram NR, Knowles JD, Oates MJ (2001) PESA-II: region-based selection in evolutionary multiobjective optimization. In: Proceedings of the GECCO, pp 283–290

Delattre M, Hansen P (1980) Bicriterion cluster analysis. IEEE Trans Pattern Anal Mach Intell 2(4):277–291

Falkenauer E (1998) Genetic algorithms and grouping problems. Wiley, New York

Handl J, Knowles J (2005) Exploiting the trade-off – the benefits of multiple objectives in data clustering. In: Evolutionary multi-criterion optimization. Springer, Berlin/Heidelberg, pp 547–560

Handl J, Knowles J (2007) An evolutionary approach to multiobjective clustering. IEEE Trans Evol Comput 11(1):56–76

Handl J, Knowles J (2013) Evidence accumulation in multiobjective data clustering. In: Evolutionary multi-criterion optimization. Springer, Berlin/Heidelberg, pp 543–557

Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. ACM Comput Surv 31(3): 264–323

Jones DR, Beltramo MA (1991) Solving partitioning problems with genetic algorithms. In: Belew RK, Booker LB (eds) Proceedings of the fourth international conference on genetic algorithms. Morgan Kaufmann, pp 442–449

Liu Y, Ram S, Lusch RF, Brusco M (2010) Multicriterion market segmentation: a new model, implementation and evaluation. Mark Sci 29(5): 880–894

Park Y-J, Song M-S (1998) A genetic algorithm for clustering problems. In: Proceedings of the third annual conference on genetic programming. Morgan Kaufman, pp 568–575

Pizzuti C (2012) A multiobjective algorithm to find communities in complex networks. IEEE Trans Evol Comput 16(3):418–430

Tibshirani R, Walther G, Hastie T (2001) Estimating the number of clusters in a dataset via the Gap statistic. J R Stat Soc: Ser B (Stat Methodol) 63(2): 411–423

Evolutionary Computation

- ▸ Evolutionary Algorithms

Evolutionary Computation in Economics

Biliana Alexandrova-Kabadjova[1], Alma Lilia García-Almanza[2], and Serafín Martínez-Jaramillo[3]
[1]Banco de México, Mexico City, Mexico
[2]Directorate of Regulation and Supervision, Banco de México, Mexico City, Mexico
[3]Directorate of Financial System Risk Analysis, Banco de México, Mexico City, Mexico

Definition

Evolutionary computation (EC) is a field in computational intelligence that takes its inspiration from nature to develop methods that resolve continuous optimization and combinatorial optimization problems. When it comes to economics, it is the area of research that involves the use of EC techniques, also subclassified as evolutionary algorithms (EAs), cultural algorithms, and self-organization algorithms, among others, in order to approach topics in economic science. The algorithms, defined as generic population-based metaheuristic optimization algorithms, are developed on the basis of the concept of biological evolution and use iterative processes such as reproduction, mutation, recombination, and selection. Some of these methods, such as genetic algorithms (GAs), genetic programming (GP), evolutionary programming (EP), estimation of distribution algorithms (EDA), evolutionary strategies (ESs), memetic algorithms, harmony search, and artificial life, have been studied and applied in computer science for more than 50 years. In mainstream economics, even though we can track the early application of GAs in game theory to as long ago as 30 years, the adoption of these

methods has been slow. This area of knowledge is different from the field of evolutionary economics, which does not necessarily apply EC techniques to the study of economic problems. The use of EC in economics pursues different aims. One is to overcome some of the limitations of classical economic models and to loosen some of the strong assumptions such models make.

Motivation and Background

EC techniques, among many other machine-learning techniques, have proven to be quite flexible and powerful tools in many fields and disciplines, such as computational linguistics, computational chemistry, and computational biology. Economics-affiliated fields are by no means the exception for the widespread use of these evolutionary-inspired methods. In addition to the undeniable necessity of computing in almost every aspect of our modern lives, numerous problems in economics possess an algorithmic nature. Economists should consider computational complexity to be an important analytical tool due to the fact that some of such problems belong to the class of NP-complete (The NP-complete computational complexity class is a subset of harder problems within the NP computational class, which is the set of all the decision problems which can be solved using a nondeterministic Turing machine in polynomial time (Papadimitriou 1994).) problems. This having been said, EC has been intensively used as an alternative approach to analytical methods used to tackle numerous NP-complete problems with considerable success, mainly in the areas of game theory, econometrics, and agent-based economic modeling. Game theory is a branch of applied mathematics that attempts to model an individual's strategic behavior. The first study considered to establish the fundamentals of the field is the book Theory of Games and Economic Behavior (John von and Oskar 1944). The idea behind this theory is that the success of an individual's decisions depends on the decisions of others. Whereas originally, the aim of the theory was to study competition, in which one agent does better at another expense (zero-sum games), now it has been extended to the study of a wider class of interactions among individuals. Furthermore, it is used extensively in economics, biology, and political science, among other disciplines.

The first work in economics (The first such work approached a classic game known as the prisoner's dilemma.) that involved the use of EC dates to the 1980s. In Robert and Hamilton (1981) and Robert (1987) the authors used GAs to derive strategies for the Iterated Prisoner's Dilemma (IPD). From then on, EC techniques in economics were used in areas such as macroeconomics, econometrics, game theory, auctions, learning, and agent-based models. There is even a school of thought in economics known as evolutionary economics (See, for example, Ulrich (2008) for an introduction.) in which the approach to the study of economics involves concepts in evolution but does not necessarily rely on EC techniques.

Econometrics is a field within the wider area of economics which involves the use of statistics and its tools to measure relationships postulated by economic theory (William 2003). In particular, it is applied to macroeconomic analysis to make out the relationships between the aggregated variables that explain broad sectors of an economy. One of the first applications of GP to econometrics was made by the creator of GP himself in John (1992).

Regarding agent-based computational economics, this field can be thought of as a branch of a wider-area, agent-based modeling (Wooldridge 2002). The field of agent-based modeling is not restricted to economics. It has been applied to social sciences in general (Robert 2003), to some classical and not so classical problems in computer science, and in some other disciplines. Axelrod provides an account of his experience using agent-based methodology for several problems, and he suggests that agent-based modeling can be seen as a bridge between disciplines. Axelrod and Tesfatsion provide a

good guide to the literature relevant to agent-based modeling in Robert and Leigh (2006). In Shu-Heng (2007) there is a thorough introduction to agents in economics and finance. In this work, Chen conceives of the agents not just as economic agents but also as computational intelligent units.

Structure of the Evolutionary Computation in Economics

The main areas addressed by EC in economics are game theory, econometrics and economic models, and agent-based economic modeling. In game theory, a well-defined mathematical object, the game, consists of a set of players and a set of strategies (decisions) available to those players. In addition, for each combination of strategies, specification of payoffs is provided. The aim of traditional applications of game theory was to find a Nash equilibrium, a solution concept, in which each player of the game adopts a strategy that is unlikely to be changed. This solution concept was named after John Nash, whose work was published in the early 1950s (John 1950). Nevertheless, it took almost 20 years to fully realize what a powerful tool Nash had created. Nowadays, game theory is one of the best established theories in economics, and it has been used extensively to model interactions among economic agents. However, games typically have many Nash equilibria, and one key assumption is that the agents behave in a rational way. In more realistic games, the equilibrium selection problem does not have an easy solution. Human behavior observed in real life, indeed, is frequently irrational. Given these constraints, evolutionary game theory was proposed as an application of the mathematical theory of games to biological contexts. In this field, Maynard Smith is considered to be the first to define the concept of an evolutionary stable strategy in John Maynard (1972). Furthermore, the possibility of using computer modeling as an extension of game theory was first explored in Robert and Hamilton (1981). Since then, computer science has been used in traditional game theory problems, like the strategic behavior of agents in auctions, auction mechanism design, etc. By providing approximate solutions to such complex problems, this approach can be useful where analytical solutions have not been found. For instance, the iterative prisoner's dilemma is one of the games most studied by researchers from computer science (Robert 1987). The prisoner's dilemma is a classic game that consists of the decision-making process for two prisoners who can choose to cooperate or defect from a group. In the case that the two prisoners choose to cooperate, they get a payoff of three each. In the case that both choose to defect, they get a payoff of one each, and in the case that one decides to defect and the other to cooperate, the former gets a payoff of five and the later a payoff of zero. In equilibrium, both players decide to defect despite the fact that it would be better for them to cooperate.

Game theory is one of the most important areas of economics because it has applications to many other fields, such as corporate decision-making, microeconomics, market modeling, public policy analysis, and environmental systems. We can find more applications of EC to game theory than IPD. For example, other work related to game theory and EC is that done by John and Engle-Warnick (2001), which deals with the well-known two-player, repeated ultimatum game. In this work they used GP as a means of inferring the strategies that were played by subjects in economic decision-making experiments. Other research related to game theory includes the duopoly and oligopoly games (Shu-Heng and Ni 2000). References regarding cooperation, coalition, and coordination are also frequent and usually driven by EC techniques (Vriend 1995). In 2006, the authors applied GP to find strategies for sequential bargaining procedure and confirmed that equilibria can be approximated by GP. This provides an opportunity to find approximate solutions to more complex situations for which theoretical solutions have yet to be found.

Regarding econometrics, in Adriana and Alexandr (2001) the authors use GAs and

simulated annealing (SA) for econometric modeling; they found that the performance of the evolutionary algorithms (EAs) is better than the performance of traditional gradient techniques on the specific models in which they performed the comparison. Finally, Ralf Östermark (1999) uses a hybrid GA in several ill-conditioned econometric and mathematical optimization problems with good results.

In addition to the use of EC in econometrics, some classical economic models like the cobweb model and exchange-rate models have been approached with EC techniques. For instance, in Jasmina (1994) and Shu-Heng and Chia-Hsuan (1996), in the former work, the author uses GAs to approach the cobweb model, whereas in the latter the authors use GP. Furthermore, Arifovic explores the use of GAs in foreign exchange markets in Jasmina (1996). The GA mechanism elaborated in such works developed decision rules that were used to determine the composition of agents' portfolios in a foreign exchange market. Arifovic made two observations rarely seen in the standard overlapping generations (OLG) model with two currencies. First, she noted that the returns and exchange rates were generated endogenously and, second, that the models' equilibrium dynamics were not stable and showed bounded oscillations (the theoretical model implies a constant exchange rate).

The use of GAs in economic modeling is not restricted to the abovementioned works. In James et al. (1995), the authors studied a version of the growth model in which physical capital is accumulated in a standard form, but human capital accumulation is subject to increasing returns. In their model, the agents make two decisions when they are young: how much to save by renting physical capital to the companies and how much to invest in training. Returns on training depend on the average level of human capital in the economy. The authors introduce agents' learning by means of GAs. In 1990, Marimon develops an economic model in which the agents adapt by means of a GA.

The final approach is agent-based computational models built with EAs for applications in economics (ACEs). In ACEs, one of the main goals is to explain the macro-dynamics of an economy by means of the micro-interactions of the economic agents. This approach to the study of an economy has been called a bottom-up approach in contrast to more traditional approaches. An additional purpose of ACEs is to handle real-world issues, something now possible due to technological advances in computational tools.

Nevertheless, to achieve a realistic representation of the agent in a model allows us to start with a critical revision of the assumptions behind classical economic theory. One of the most important concepts in this context is rationality. It is at the core of most economic models. It is frequently assumed that economic agents behave in a fully rational way. Unfortunately, it is not clear what this assumption holds, especially in view of irrational behavior observed during recurrent financial crises.

Herbert A. Simon is probably the best known scientist to claim that "decision-making" under uncertainty is not a fully rational process. He developed his theory based on the concept of bounded rationality (Herbert 1957). He was one of the pioneers in the field of artificial intelligence (AI), as well as a highly respected psychologist and economist. Later, in Brian (1991), the author made important contributions to the development of agents with bounded rationality using computational tools. Some more recent ideas about rationality from a computer scientist's point of view are found in Edward (2008).

Some other common assumptions behind classical economic theory are that the participants of the model have *homogeneous preferences* and they *interact globally* (Robert 2000). Departing from the assumption of full rationality and homogeneous expectations, the horizon and the design issues vary widely. The modeling of the learning behavior of the agents is a central part of the research agenda in computational economics. Regarding the agents' learning process, Lucas' definition for adaptive behavior from the economic point of view is of extreme importance (Robert 1986). There are many useful techniques to implement this adaptive learning. The application of genetic algorithms

(GAs) in James and John (1999) and genetic programming (GP) in Serafin and Edward (2009) are good examples. GP has been previously described as a suitable way to model economic learning in Bruce (1999). In Thomas (2006), the author provides us a summary of the available options to model agent behavior and learning in economics.

With the use of programming languages, the agent-based approach allows us to represent explicitly agents with bounded rationality and heterogeneous preferences. Given a specific social structure, the simulation of the interaction among agents is the strength and heart of agent-based modeling (ABM). Nowadays ABM is a promising area of research, which has opened the way to social scientists to look for new insights in resolving important real-world issues. Considered the third way of doing science (Robert 2003), modeling the behavior of the autonomous decision-making entities allows researchers to simulate the emergence of certain phenomena in order to gain better understanding of the object of study (Robert 2000). In this sense ACE, defined as the computational study of economic processes modeled as dynamic systems of interacting agents (Leigh 2006), is a growing area inside the field of agent-based modeling. ACE research is developing rapidly. By using machine-learning techniques, researchers model the agents as software programs able to make autonomous decisions. Consequently, the interactions among the individuals at the microlevel give rise to regularities at the macrolevel (globally). The intention is to observe the emerging self-organizing process for a certain period of time, in order to study the presence of patterns or the lack of them. Currently, the study of this self-organizing capability is one of the most active areas of ACE research. EAs have been used for the modeling of the agents' learning in multi-agent simulations. In economics, it is possible to find very different approaches and topics. The following is a small selection from a large body of literature:

Electricity Markets (Massoud 2002) (Learning Classifier System)

Foreign Exchange Markets (Jasmina 1994; Kiyoshi and Kazuhiro 2001) Genetic Algorithms

Payment Card Markets (Biliana et al. 2011) (Population Based Incremental Learning)

Retail Petrol Markets (Heppenstall et al. 2007) (Genetic Algorithms)

Stock Markets (Brian et al. 1997) (Learning Classifier Systems) and;

(Serafin and Edward 2009) (GP)

Cross-References

- ▸ Evolutionary Algorithms
- ▸ Evolutionary Computation in Finance
- ▸ Evolutionary Computational Techniques in Marketing
- ▸ Genetic and Evolutionary Algorithms
- ▸ Genetic Programming

Recommended Reading

Agapie A, Agapie A (2001) Evolutionary computation for econometric modeling. Adv Model Optim 3(1): 1–5

Alexandrova-Kabadjova B, Tsang E, Krause A (2011) Competition is bad for consumers: analysis of an artificial payment card market. J Adv Comput Intell Intell Inform 15:188–196

Amin M (2002) Restructuring the electric enterprise: simulating the evolution of the electric power industry with intelligent adaptive agents. In: Faruqui A, Eakin K (eds) Market analysis and resource management, chapter 3. Kluwer Academic Publishers, Boston/Dordetch/London

Arifovic J (1994) Genetic algorithm learning and the cobweb model. J Econ Dyn Control 18:3–28

Arifovic J (1996) The behavior of the exchange rate in the genetic algorithm and experimental economics. J Political Econ 104:510–541

Arthur WB (1991) Learning and adaptive economic behavior. Designing economic agents that act like human agents: a behavioral approach to bounded rationality. Am Econ Rev 81:353–359

Arthur WB, Holland JH, LeBaron B, Palmer RG, Talyer P (1997) Asset pricing under endogenous expectations in an artificial stock market. In: Brian Arthur W, Durlauf S, Lane D (eds) The economy as an evolving complex system II. Addison-Wesley, Reading

Axelrod R (1987) The evolution of strategies in the iterated prisoner's dilemma. Genetic algorithms and simulated annealing of research notes in AI, chap-

ter 3. Pitman/Morgan Kaufmann, London/Los Altos, pp 32–41
Axelrod R (2003) Advancing the art of simulation in the social sciences. Jpn J Manag Inf Syst, Spec Issue Agent-Based Model 12(3):16–22
Axelrod R, Hamilton WD (1981) The evolution of cooperation. Science 211:1390–1396
Axelrod R, Tesfatsion L (2006) A guide for newcomers to agent-based modeling in the social sciences. In: Judd KL, Tesfatsion L (eds) Handbook of computational economics, volume 2: agent-based computational economics. Handbooks in economics, chapter Appendix A. North-Holland Amsterdam, pp 1647–1656
Axtell R (2000) Why agents? On the varied motivations for agent computing in the social sciences. Working paper 17, Center on Social and Economic Dynamics
Brenner T (2006) Agent learning representation advice in modelling economic learning. In: Judd KL, Tesfatsion L (eds) Handbook of computational economics, volume 2: agent-based computational economics. Handbooks in economics, chapter 18. North-Holland, pp 895–948
Bullard J, Arifovic J, Duffy J (1995) Learning in a model of economic growth and development. Working paper 1995-017A, Federal Reserve Bank Of St. Louis
Bullard J, Duffy J (1999) Using genetic algorithms to model the evolution of heterogeneous beliefs. Comput Econ 13:41–60
Chen S-H (2007) Editorial: computationally intelligent agents in economics and finance. Inf Sci 177(5):1153–1168
Chen S-H, Ni CC (2000) Simulating the ecology of oligopolistic competition with genetic algorithms. Knowl Inf Syst 2(2):285–309
Chen S-H, Yeh C-H (1996) Genetic programming learning in the cobweb model with speculators. In: International computer symposium (ICS'96). Proceedings of international conference on artificial intelligence. National Sun Yat-Sen University, Kaohsiung, R.O.C., 19–21, pp 39–46
Duffy J, Engle-Warnick J (2001) Using symbolic regression to infer strategies from experimental data. In: Chen S-H (ed) Evolutionary computation in economics and finance. Physica-Verlag, New York, pp 61–82
Edmonds B (1999) Modelling bounded rationality in agent-based simulations using the evolution of mental models. In: Brenner T (ed) Computational techniques for modelling learning in economics. Kluwer, Boston, pp 305–332
Greene WH (2003) Econometric analysis, 5th edn. Prentice Hall, Upper Saddle River, 07456
Heppenstall A, Evans A, Birkin M (2006) Using hybrid agent-based systems to model spatially-influenced retail markets. J Artif Soc Soc Simul 9(3): 2
Izumi K, Ueda K (2001) Phase transition in a foreign exchange market-analysis based on an artificial market approach. IEEE Trans Evol Comput 5(5):456–470
Jin N, Tsang EPK (2006) Co-adaptive strategies for sequential bargaining problems with discount factors and outside options. In: Proceedings of the IEEE congress on evolutionary computation, Vancouver. IEEE Press, pp 7913–7920
Koza J (1992) A genetic approach to econometric modelling. In: Bourgine P, Walliser B (eds) Economics and cognitive science. Pergamon Press, Oxford/New York, pp 57–75
Lucas RE (1986) Adaptive behavior and economic theory. In: Hogarth RM, Reder MW (eds) Rational choice: the contrast between economics and psychology. University of Chicago Press, Chicago/London, pp 217–242
Marimon R, McGrattan E, Sargent TJ (1990) Money as a medium of exchange in an economy with artificially intelligent agents. J Econ Dyn Control 14:329–373
Martinez-Jaramillo S, Tsang EPK (2009) An heterogeneous, endogenous and coevolutionary gp-based financial market. IEEE Trans Evol Comput 13:33–55
Nash J (1950) The barganing problem. Econometrica 18:155–162
Östermark R (1999) Solving irregular econometric and mathematical optimization problems with a genetic hybrid algorithm. Comput Econ 13(2): 103–115
Papadimitriou C (1994) Computational complexity. Addison-Wesley, Reading
Simon HA (1957) Models of man: social and rational. John Wiley and Sons, Inc., New York
Smith JM (1972) Game theory and the evolution of fighting. Edinburgh University Press, Edinburgh, pp 8–28
Tesfatsion L (2006) Agent-based computational economics: a constructive approach to economic theory. In: Judd KL, Tesfatsion L (eds) Handbook of computational economics, volume 2: agent-based computational economics. Volume 2 of handbooks in economics, chapter 16. North-Holland Amsterdam, pp 831–880
Tsang EPK (2008) Computational intelligence determines effective rationality. Int J Autom Comput 5:63–66
von Neumann J, Morgenstern O (1944) Theory of games and economic behavior. Princeton University Press, Princeton
Vriend NJ (1995) Self-organization of markets: an example of a computational approach. Comput Econ 8: 205–231
Witt U (2008) Evolutionary economics. In The New Palgrave Dictionary of Economics. Second Edition. Eds. Steven N. Durlauf and Lawrence E. Blume. Palgrave Macmillan, London
Wooldridge M (2002) An introduction to multiAgent systems. Wiley, Chichester

Evolutionary Computation in Finance

Serafín Martínez-Jaramillo[1], Tonatiuh Peña Centeno[2], Biliana Alexandrova-Kabadjova[3], and Alma Lilia García-Almanza[4]
[1]Directorate of Financial System Risk Analysis, Banco de México, Mexico City, Mexico
[2]German Center for Neurodegenerative Diseases, Banco de México, Mexico City, Mexico
[3]Banco de México, Mexico City, Mexico
[4]Directorate of Regulation and Supervision, Banco de México, Mexico City, Mexico

Definition

Evolutionary computation (EC) in finance is an area of research and knowledge which involves the use of EC techniques in order to approach topics in finance. This area of knowledge is similar to EC in economics; in fact, the areas frequently overlap in some of the topics they approach. The application of EC in finance pursues two main purposes: first, to overcome the limitations of some theoretical models, also departing from some of the assumptions made in those models, and, second, to innovate in this extremely competitive area of research, given the powerful economic incentives to do so.

EC techniques have been widely used in a variety of topics in finance. Among the most relevant we find: financial forecasting, algorithmic and automatic trading, option pricing, portfolio optimization, artificial financial markets, credit rating, credit scoring, bankruptcy prediction, and filtering techniques.

The views expressed here are those of the authors and do not represent the views of the Mexican central bank. The authors are grateful with Alberto Romero Aranda and Dorothy Walton for their valuable comments on this entry.

Motivation and Background

Evolutionary computation (EC) is a field in machine learning (ML) in which the techniques developed apply the principle of evolution in different ways. Among the many techniques which have been used in financial applications, one can find genetic algorithms (GAs), genetic programming (GP), learning classifier systems (LCSs), population-based incremental learning (PBIL), grammatical evolution (GE), evolutionary strategies (ESs), memetic algorithms (MAs), and evolutionary nearest neighbor classifier algorithm (ENPC), among many others. In addition, many of the above mentioned techniques are used in combination or as meta-techniques on top of other machine-learning tools. In many financial markets, competition is at the center of everyday activities undertaken by individuals and companies. As a consequence, given this fierce competition and the necessity for innovation, it is natural to find numerous problems in finance being approached by existing EC techniques. For example, in stock markets, individual and institutional investors try to beat the market in order to make more profits than other market participants. Coming up with novel algorithms or techniques is crucial to maintaining their performance and status in relation to competitors.

This area of research has been given many different names, including computational finance and computational intelligence in finance, among others. Research in this area is still evolving. Therefore, it is difficult to define the field clearly or to establish its limits. Moreover, nowadays it is almost impossible to provide a full account of all the relevant work that involves any form of EC in finance. It is also hard to organize the vast amount of human knowledge implicit in the field. The number of specialized journals, meetings, and books is indeed very large and getting larger. Chen (2002a), Chen and Wang (2004), and Chen et al. (2007) exemplify important research in this dynamic field.

Computing in finance is an almost unavoidable tool, from Monte Carlo simulation to computer intensive methods used to price complex derivatives. Furthermore, some of the most crit-

ical processes in finance make heavy use of computers. Computational finance is a frequently mentioned term, sometimes associated with financial engineering. However, in this context we refer to computational finance as the use of non-conventional computational techniques, like EC or other machine-learning techniques, to tackle problems in finance. See, for example, Tsang and Martinez-Jaramillo (2004) for a good introduction to the field. Additionally, Chen (2002b), Brabazon and O'Neill (2008), and Brabazon and O'Neill (2009) illustrate relevant works in the field.

Financial Forecasting and Algorithmic and Automatic Trading

In recent years, computers have shown themselves to be a powerful tool in financial applications. For that reason, many machine-learning techniques have been applied to financial problems. Financial forecasting is one of the most important fields in the area of computational finance (Tsang and Martinez-Jaramillo 2004). EC has been used to solve a great variety of financial forecasting problems, such as prediction of stock prices changes and their volatility, forecasting in foreign exchange markets, and more. Let us introduce some of the most important research in the financial forecasting area. This does not pretend to be either an extensive or detailed survey of literature in the field. The objective is just to illustrate the use of EC in financial forecasting applications.

Machine-learning classifiers, like other forecasting techniques, extend past experiences into the future. The aim is to analyze past data in order to identify patterns in the interest of creating a model or a set of rules to predict future events. In particular, EC techniques have some characteristics that make them useful for financial forecasting. For example, evolutionary techniques are able to produce interpretable solutions. This characteristic is especially important for predictions, since the main goals of classification are to (1) generate an accurate classification model that should be able to predict unseen cases and (2) discover the predictive structure of a problem (Breiman et al. 1984).

Models which help to understand the structural patterns in data provide information that can be useful for recognizing the variables' interactions. There are classification models that have good predictive power. However, these models provide a poor representation of the solution (take, e.g., the artificial neural networks). Since EC techniques provide not just good predictions but interpretable solutions, they have been used in financial problems to acquire knowledge of the event to predict. For example, Tsang et al. (2004) trained a GP using past data from the financial stock markets to predict price movements of at least $r\,\%$ within a period of at most n time units. The attributes used to train the GP were indicators from technical analysis. Due to the possibility of interpreting the solution, the authors were able to analyze the most successful indicators in the result. In fact, some researchers have used EC in order to discover new financial indicators. This include Allen and Karjalainen (1999), who made use of a GP system to infer technical trading rules from past prices. The algorithm was applied to the S&P 500. Bhattacharyya et al. (2002) used GP to discover trading decision models from high-frequency foreign exchange (FX) market data.

In other related works, Bhattacharyya et al. (2002) used GA for mining financial time series to identify patterns, with the aim of discovering trading decision models. Potvin et al. (2004) applied GP to automatically generate short-term trading rules on the stock markets. The authors used historical pricing and transaction volume data reported for 14 Canadian companies from the Toronto Stock Exchange market. Another approach called grammatical evolution (GE) (Brabazon and O'Neill 2004) was applied to discover new technical trading rules, which can be used to trade on foreign exchange markets. In that approach, each of the evolved programs represents a market trading system.

Additionally, EC techniques are able to generate a set of solutions for a single problem. This characteristic has been used to obtain a set

of results with the aim of applying the most suitable solution to the particular problem. For instance, Lipinski (2004) analyzed high-frequency data. The independent variables were composed by 350 expert rules and observations of stock price quotations and order books recorded from the Paris Stock Exchange. In that model, stock market trading rules were combined into stock market trading experts, which defined the trading expertise. The author used a simple GA, a population-based incremental learning (PBIL), a compact genetic algorithm (CGA), and an extended compact genetic algorithm (ECGA) to discover optimal trading experts in a specific situation. The author argues that the optimal solution depends on the specific situation in the stock market, which varies with time. Thus, optimal trading experts must be rebuilt. EC plays an important role in learning and continual adaptation to the changing environment.

Taking advantage of the EC's ability to generate multiple solutions, Garcia-Almanza and Tsang (2008) proposed an approach, called Evolving Comprehensible Rules (ECR), to discover patterns in financial data sets to detect investment opportunities. ECR was designed to classify the minority class in unbalanced environments, which is particularly useful in financial forecasting given that very often the number of profitable opportunities is scarce. That approach offers a range of solutions to suit an investor's risk guidelines. Thus, the user can choose the best trade-off between misclassification and false alarm costs according to the investor's requirements. The approach proposed by Ghandar et al. (2008) was designed to generate trading rules. The authors implemented an adaptive computational intelligent system by using an evolutionary algorithm and a fuzzy logic rule-based representation. The data used to train the system was composed just of volume and price. The authors' objective was to create a system to generate rules to buy recommendations in dynamic market conditions. An analysis of the results was provided by applying the system for portfolio construction to historical data for companies listed on the MSCI Europe Index from 1990 to 2005. The results showed that their approach was able to generate trading rules that beat traditional fixed rule-based strategies, such as price momentum and alpha portfolios, and the approach also beat the market index.

Given that EC can be used as an optimization technique, EC techniques have been combined with other approaches. For example, Chen et al. (1999) used a genetic algorithm to determine the number of input variables and the number of hidden layers in an NN for forecasting Dollar/Deutsche mark foreign exchange rates. Chen and Lu (1999) used GP to optimize a NN. That approach is called evolutionary neural trees (ENTs). The objective was to forecast the high-frequency stock returns of the Heng Seng stock index. Schoreels et al. (2004) investigated the effectiveness of an agent-based trading system. The system employs a simple GA to optimize the trading decisions for every agent; the knowledge is based on a range of technical indicators generating trading signals. In Dempster et al. (2001) the authors aim to detect buy and sell signals in the FX markets. The authors analyze and compare the performance of a GP combined with a reinforcement learning (RL) system to a simple linear program (LP) characterizing a Markov decision process (MDP) and a heuristic in high-frequency (intraday) FX trading. The authors consider eight popular technical indicators used by intraday FX traders based on simple trend indicators such as moving averages as well as more complex rules. From experimental results, the authors found that all methods were able to create significant in-sample and out-of-sample profits when transaction costs are zero. The GP approach generated profits for nonzero transaction costs, although none of the methods produce significant profits at realistic transaction costs.

As is evident, EC techniques allow the representation of solutions using different structures, such as decision trees (Potvin et al. 2004), finite-state automata, graphs, grammar (Brabazon and O'Neill 2004), networks, and binary vectors (Lipinski 2004), among many others. This characteristic lets us choose the best representation for the problem.

Portfolio Optimization

Portfolio optimization is an all-important field in finance. The portfolio selection problem can be described in a simple way as the problem of choosing the assets and the proportion of such assets in an investor's wealth in an effort to maximize profits and minimize risk.

As the name suggests, *portfolio optimization* is an optimization problem and EC has proven to be very useful in difficult (sometimes intractable) optimization problems. In Maringer (2005), the author explains extensively the portfolio optimization problem and the possible heuristic approaches, including Ant Systems (AS), memetic algorithms (MAs), genetic algorithms (GAs), and evolutionary strategies (ESs). For an extensive review from a financial economic perspective, see Brandt (2009).

Being a multi-objective optimization problem, EC provides plenty of opportunities to approach the portfolio optimization problem. For example, Hassan and Clack (2008) uses a multi-objective GP to approach this problem. In Diosan (2005), the author compares different multi-objective evolutionary algorithms for the portfolio optimization problem.

The number of papers on portfolio optimization using machine-learning techniques is large. Streichert et al. (2004), Doerner et al. (2004), and Maringer (2006) are some significant works on portfolio optimization that use some form of evolutionary computation or artificial intelligence.

Multi-objective evolutionary optimization is an important field within EC, and the portfolio optimization problem is not the only application in finance which can be approached. In Coello (2006), the author surveys the literature on multi-objective optimization in economics and finance.

Financial Markets

This section introduces the applications of EC in artificial financial markets. Due to the extensiveness of the literature, only a general overview will be provided. For a more complete and detailed guide to the applications of EC techniques in artificial financial markets, see Martinez-Jaramillo and Tsang (2009a).

Financial markets are essential for financial systems. Such markets represent one of the most efficient ways to allocate financial resources to companies. However, bubbles and crashes are recurrent phenomena which have enormous repercussions for the global economy. Indeed, nowadays we can see as never before that one single crash in one market can lead to a worldwide slump on most of the other stock markets. Moreover, crisis in financial markets can affect other aspects of the (real) economy, for example, interest rates, inflation, unemployment, etc. This, in turn, can cause even more instability on the financial markets.

Financial markets are very important in our lives, whether we like it or not. For example, everyone suffers the consequences of a stock market crash such as the international market crash in 1987. Moreover, this phenomena (market crashes) occurs with an unpleasantly higher frequency than predicted by standard economic theory. Important references on rare disasters and asset markets are Barro (2009), Gabaix (2012), and Gourio (2008). One of the most important research issues in financial markets is an explanation for the process that determines asset prices and, as a result, rates of return. There are many models that can be used to explain such processes, such as the capital asset pricing model (CAPM) (Sharpe 1964), arbitrage pricing theory (APT) (Ross 1976), or Black-Scholes option pricing (Black and Scholes 1973).

Nevertheless, financial markets are very complex to analyze due to the wide variety of participants and their ever-changing nature. The most common approach to study them is by means of analytical models. However, such models have some limitations which, in turn, have led to the search for alternative methods to approach them. Agent-based computational economics (ACE) (Tesfatsion 2002) and computational finance (Tsang and Martinez-Jaramillo 2004) have risen as alternative ways to overcome some of the problems of the analytical models.

Agent-based financial markets with varying characteristics have been developed for the study of such markets in the last decade, since the

influential Santa Fe Artificial Market (The Santa Fe Artificial Stock Market is a simulated stock market developed at the Santa Fe Institute. The market was developed by a team of highly regarded researchers, among them is John Holland, the inventor of genetic algorithms Holland 1975.) (Arthur et al. 1997). Some of them differ from the original Santa Fe market in the type of agents used, such as Chen and Yeh (2001), Gode and Sunder (1992), Yang (2002), and Martinez-Jaramillo and Tsang (2009b), and in market mechanisms, such as Bak et al. (1997), Gode and Sunder (1992), and Yang (2002). Other markets borrow ideas from statistical mechanics, such as Levy et al. (1994) and Lux (1998). Some important research has been done modeling stock markets inspired by the minority game (The minority game was first proposed by Yi-Cheng Zhang and Damien Challet (1997) inspired by the El Farol bar problem introduced by Brian Arthur 1994.) like Challet et al. (2000). There are financially simulated markets in which several stocks are traded, such as in Cincotti et al. (2005). However, criticism of this approach centers on the problem of calibration, the numerous parameters needed for the simulation program, and the complexity of simulation, among other problems. The contradictions between existing theory and the empirical properties of stock market returns are the main driving force for some researchers to develop and use different approaches to study financial markets. An additional aspect of the study of financial markets is the complexity of the analytical models of such markets. Prior to the development of some new simulation techniques, very important simplifying (unrealistic) assumptions had to be made in order to allow for the tractability of the theoretical models.

Artificial intelligence and, in particular, EC have been used in the past to study financial and economic problems. However, the development of a well-established community known as the agent-based computational economics community facilitates the study of phenomena in financial markets that was not previously possible. Within this community, a vast number of studies and approaches are being produced in order to solve or gain more understanding of economic problems.

The influential study (Arthur et al. 1997) and previously the development of the concept of bounded rationality in Simon (1982) and Arthur (1991) changed the way in which we conceive and model economic agents. This change in conception dramatically altered the possibilities for studying some economic phenomena and, in particular, financial markets. The new models of economic agents have changed. There is no longer any need for fully rational representative agents or for homogeneous expectations and information symmetry. Furthermore, the development of artificially adapted agents (Holland and Miller 1991) provides a way forward for economic science to study economic systems.

Although they all differ in the sorts of assumptions made, methodology, and tools, these markets share the same essence: the macrobehavior of such markets (usually the price) should emerge endogenously as a result of the microinteractions of the (heterogeneous) market participants. This approach is in opposition to traditional techniques used in economics and finance. Moreover, in Lux and Ausloos (2002) the authors declare:

> Unfortunately, standard modelling practices in economics have rather tried to avoid heterogeneity and interaction of agents as far as possible. Instead, one often restricted attention to the thorough theoretical analysis of the decisions of one (or few) *representative* agents.

The *representative agent* is a common, yet very strong, assumption in the modeling of financial markets. This concept has been the source of controversy and strong criticism. For example, in Kirman (1992), the author criticizes the *representative individual* approach in economics.

In order to understand the approaches in artificial (simulated) financial markets, it is useful to describe the different types of markets on the basis of the framework proposed in LeBaron (2001). In this study, LeBaron identifies the key design issues present in every artificial financial market and describes some of the most important

studies up to then. In LeBaron (2006), LeBaron surveys again the literature existing until then. The main design issues identified in LeBaron (2001) are:

- Agents
- Market mechanisms
- Assets
- Learning
- Calibration
- Time

In addition to describing the different approaches in artificial financial markets by using the above-described framework, there is a fairly detailed extension of it in Grothmann (2002) that is worth looking at. In this study, the basic design issues proposed in LeBaron (2001) are extended and given more detail.

Option Pricing

Derivatives (See Hull (2008) for an introduction to derivatives.) are financial instruments whose main purpose is to hedge risk. However, derivatives can also be used to speculate with very negative effects on the financial health of companies, as we all know now. Derivative markets have seen significant expansion in recent years. Futures, forwards, swaps, and options are the best known types of derivatives. Option pricing is an extremely important task in finance. The Black-Scholes model for option pricing is the reference analytical model since it has an important theoretical framework behind it. However, in practice, prices deviate from the prices obtained with this model. One possible reason for the departure is the assumptions being made in the model (the assumption of constant volatility and the assumption that prices follow a geometric Brownian motion). This is why GP was used as an alternative to perform option pricing in Chen et al. (1998), Chidambaran et al. (2002), Fan et al. (2007), and Yin et al. (2007). Interestingly, not only has GP been used to perform option pricing, but also ant colony optimization (ACO) has been explored to approach this important problem in finance (Kumar et al. 2008).

Credit Rating, Credit Scoring, and Bankruptcy Prediction

Credit rating and credit scoring are two examples of financial problems that have been traditionally approached through statistical analysis. A credit rating is an estimate of a corporation's worthiness to be given a credit and is generally expressed in terms of an ordinal value. Credit scoring is a technique used to express the potential risk of lending money to a given consumer in terms of a probability measure. Both techniques are similar in their ends but applied to different domains.

The seminal work in the field of credit scoring is that of Altman (1968), who proposed the application of linear discriminant analysis (Fisher 1936) to a set of measurements known as financial ratios, i.e., indicators of a corporation's financial health obtained from the corporation's financial statements. One of the main applications of Altman's method, also known as the Z-score, is bankruptcy prediction. Understandably, a series of improvements have been achieved by means of applying more powerful classifiers, such as decision trees, genetic programming, neural networks, and support vector machines, among others. References that apply such techniques or conduct a review of the literature on their application are Atiya (2001), Sung et al. (1999), West (2000), Ong et al. (2005), Shin and Lee (2002), Martens et al. (2007), and Huang et al. (2007).

Another method to evaluate credit worthiness is that provided by specialized agencies. The so-called credit ratings are nothing more than ordinal values expressing the financial history, current assets, and liabilities of entities such as individuals, organizations, or even sovereign countries, such that they represent the likelihood of default on any type of debt. Although each rating agency uses its own methodology and scale and these are usually not disclosed, in the academic realm, nevertheless, several superseding techniques to ordinal regression have been applied. For example, Huang et al. (2004), Dutta and Shekhar (1988), Paleologo et al. (2009), and Zhou et al. (2006) have proposed computationally oriented methods to solve this problem.

Related to bankruptcy prediction, NNs have been the standard selection apart from the tradi-

tional statistical methods (discriminant analysis, logit and probit models). Quintana et al. (2008) explore the feasibility of using the evolutionary nearest neighbor classifier (ENPC) algorithm suggested by Fernández and Isasi (2004) in the domain of early bankruptcy prediction. They assess its performance comparing it to six alternatives; their results suggest that this algorithm might be considered as a good choice. Another relevant study is Turku et al. (1996) in which the authors compare discriminant analysis, logit analysis, and GAs for the selection of the independent variables used for the prediction model.

Filtering Techniques

Many real-life problems involve the estimation of unknown data from observed (probably noisy) values. Direct estimation methods like the Markov chain Monte Carlo (Andrieu et al. 2003), the sequential Monte Carlo (Doucet et al. 2001), and the particle filter (Gordon et al. 1993) methods are very useful for this task. In addition to the many applications of filtering techniques, filters are also very important tools in finance and economics. Their applications in the fields of macroeconomics, microeconomics, and finance are numerous. To enumerate them all is beyond the scope of this entry.

Among the many variations of filtering techniques, the Kalman filter (Kalman 1960), the extended Kalman filter (Jazwinski 1970), the unscented Kalman filter (Julier and Uhlmann 1997), the particle filter (Gordon et al. 1993), and the hidden Markov model (Baum et al. 1970) are some of those which practitioners use most widely in finance. In economics the Hodrick-Prescott filter (Hodrick and Prescott 1997) is one of the most widely used.

These methods have benefited from the application of EC techniques to optimize over the parameter space and to improve the performance of the methods in particular applications. For example, in O'Sullivan (2007), the authors optimize over the parameter space by using an evolutionary optimizer known as differential evolution (DE) for a Cox, Ingersoll, and Ross term-structure model. The authors in Rezaei et al. (2008) make use of EC techniques to improve the performance of a Kalman filter by means of GAs. In Kumar et al. (2010) the authors tune an extended Kalman filter using different EAs.

In an interesting application of EC techniques in the tuning of Kalman filters, (Huo et al. 2014) determines the initial parameterization of a Kalman filter with a GA, and the parameterization is adaptively updated by means of a Fuzzy Inference System (FIS).

Cross-References

- ▸ Evolutionary Algorithms
- ▸ Evolutionary Computation in Economics
- ▸ Evolutionary Computational Techniques in Marketing
- ▸ Genetic Programming

Recommended Reading

Allen F, Karjalainen R (1999) Using genetic algorithms to find technical trading rules. J Financ Econ 51:245–271

Altman EI (1968) Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. J Financ 23(4):589–609

Andrieu C, de Freitas N, Doucet A, Jordan MI (2003) An introduction to MCMC for machine learning. Mach Learn 50:5–43

Arthur WB (1991) Learning and adaptive economic behavior. Designing economic agents that act like human agents: a behavioral approach to bounded rationality. Am Econ Rev 81:353–359

Arthur WB (1994) Inductive reasoning and bounded rationality: the El Farol problem. Am Econ Rev 84:406–411

Arthur WB, Holland JH, LeBaron B, Palmer RG, Talyer P (1997) Asset pricing under endogenous expectations in an artificial stock market. In: Arthur WB, Durlauf S, Lane D (eds) The economy as an evolving complex system II. Addison-Wesley, Reading

Atiya AF (2001) Bankruptcy prediction for credit risk using neural networks: a survey and new results. IEEE Trans Neural Netw 12(4):929–935

Bak P, Paczuski M, Shubik M (1997) Price variations in a stock market with many agents. Physica A 246:430–453

Barro RJ (2009) Rare disasters, asset prices, and welfare costs. Am Econ Rev 99(1):243–264

Baum LE, Petrie T, Soules G, Weiss N (1970) A maximization technique occurring in the statistical

analysis of probabilistic functions of Markov chains. Ann Math Stat 41:164–171
Bhattacharyya S, Pictet OV, Zumbach G (2002) Knowledge-intensive genetic discovery in foreign exchange markets. IEEE Trans Evol Comput 6(2):169–181
Black F, Scholes M (1973) The pricing of options and corporate liabilities. J Political Econ 81: 637–654
Brabazon A, O'Neill M (2004) Evolving technical trading rules for spot foreign-exchange markets using grammatical evolution. Comput Manag Sci 1(3):311–327
Brabazon A, O'Neill M (eds) (2008) Natural computing in computational finance. Volume 100 of studies in computational intelligence. Springer, Berlin
Brabazon A, O'Neill M (eds) (2009) Natural computing in computational finance, vol 2. Volume 185 of studies in computational intelligence. Springer, Berlin
Brandt MW (2009) Portfolio choice problems. Handb Financ Econom 1:269–336
Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth International Group, Belmont
Challet D, Marsili M, Zhang Y-C (2000) Modeling market mechanism with minority game. Physica A 276:284–315
Challet D, Zhang Y-C (1997) Emergence of cooperation and organization in an evolutionary game. Physica A 246:407
Chen S-H (ed) (2002a) Evolutionary computation in economics and finance. Volume 100 of studies in fuzziness and soft computing. Springer, New York/Secaucus
Chen S-H (ed) (2002b) Genetic algorithms and genetic programming in computational finance. Kluwer Academic Publishers, Norwell
Chen S-H, Lu C-F (1999) Would evolutionary computation help in designs of artificial neural nets in forecasting financial time series? In: Proceeding of 1999 congress on evolutionary computation, Washington, DC. IEEE Press, pp 275–280
Chen S-H, Wang H-S, Zhang B-T (1999) Forecasting high-frequency financial time series with evolutionary neural trees: the case of hang-seng stock index. In: Arabnia HR (ed) Proceedings of the international conference on artificial intelligence, IC-AI'99, Las Vegas, vol 2, 28 June–1 July 1999. CSREA Press, pp 437–443
Chen S-H, Wang PP (eds) (2004) Computational intelligence in economics and finance. Advanced information processing. Springer, Berlin/New York
Chen S-H, Wang PP, Kuo T-W (eds) (2007) Computational intelligence in economics and finance, volume II. Advanced information processing. Springer, Berlin/Heidelberg
Chen S-H, Yeh C-H (2001) Evolving traders and the business school with genetic programming: a new architecture of the agent-based artificial stock market. J Econ Dyn Control 25(3–4):363–393
Chen S-H, Yeh C-H, Lee W-C (1998) Option pricing with genetic programming. In: Koza JR, Banzhaf W, Chellapilla K, Deb K, Dorigo M, Fogel DB, Garzon MH, Goldberg DE, Iba H, Riolo R (eds) Genetic programming 1998: proceedings of the third annual conference, University of Wisconsin, Madison, 22–25 July 1998. Morgan Kaufmann, pp 32–37
Chidambaran NK, Triqueros J, Jevons Lee C-W (2002) Option pricing via genetic programming. In: Chen S-H (ed) Evolutionary computation in economics and finance. Volume 100 of studies in fuzziness and soft computing, chapter 20. Physica Verlag, New York, pp 383–398
Cincotti S, Ponta L, Raberto M (2005) A multi-assets artificial stock market with zero-intelligence traders. In: WEHIA 2005 (13–15 June 2005), Essex
Coello CA (2006) Evolutionary multi-objective optimization and its use in finance. MIMEO, CINVESTAV-IPN, Mexico
Dempster MAH, Payne TW, Romahi Y, Thompson GWP (2001) Computational learning techniques for intraday FX trading using popular technical indicators. IEEE Trans Neural Netw 12:744–754
Diosan L (2005) A multi-objective evolutionary approach to the portfolio optimization problem. In: CIMCA'05: proceedings of the international conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce vol-2 (CIMCA-IAWTIC'06), Washington, DC. IEEE Computer Society, pp 183–187
Doerner K, Gutjahr WJ, Hart RF, Strauss C, Stummer C (2004) Pareto ant colony optimization: a metaheuristic approach to multiobjective portfolio selection. Ann Oper Res 131:79–99
Doucet A, de Freitas N, Gordon NJ (2001) An introduction to sequential Monte Carlo methods. In: Doucet A, de Freitas N, Gordon NJ (eds) Sequential Monte Carlo methods in practice. Springer, New York, pp 1–13
Dutta S, Shekhar S (1988) Bond rating: a nonconservative application of neural networks. IEEE Int Conf Neural Netw 2:443–450
Fan K, Brabazon A, O'Sullivan C, O'Neill M (2007) Option pricing model calibration using a real-valued quantum-inspired evolutionary algorithm. In: GECCO'07: proceedings of the 9th annual conference on genetic and evolutionary computation. ACM, New York, pp 1983–1990
Fernández F, Isasi P (2004) Evolutionary design of nearest prototype classifiers. J Heuristics 10(4): 431–454
Fisher RA (1936) The use of multiple measurements in taxonomic problems. Ann Eugen 7:179
Gabaix X (2012) Variable rare disasters: an exactly solved framework for ten puzzles in macro-finance. Q J Econ 127(2):645–700

Garcia-Almanza AL, Tsang EPK (2008) Evolving decision rules to predict investment opportunities. Int J Autom Comput 5(1):22–31
Ghandar A, Michalewicz Z, Schmidt M, To TD, Zurbrugg R (2008) Computational intelligence for evolving trading rules. IEEE Trans Evol Comput 13(1):71–86
Gode DK, Sunder S (1992) Allocative efficiency of markets with zero intelligence (z1) traders: market as a partial substitute for individual rationality. GSIA working papers 1992-16, Tepper School of Business, Carnegie Mellon University
Gordon NJ, Salmond DJ, Smith AFM (1993) Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In: IEE Proceedings F (Radar and Signal Processing), vol 140, IET, pp 107–113
Gourio F (2008) Disasters and recoveries. Am Econ Rev 98:68–73
Grothmann R (2002) Multi-agent market modeling based on neural networks. PhD thesis, Faculty of Economics, University of Bremen
Hassan G, Clack CD (2008) Multiobjective robustness for portfolio optimization in volatile environments. In: GECCO'08: proceedings of the 10th annual conference on Genetic and evolutionary computation. ACM, New York, pp 1507–1514
Hodrick RJ, Prescott EC (1997) Postwar us business cycles: an empirical investigation. J Money Credit Bank 29:1–16
Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
Holland JH, Miller JH (1991) Artificial adaptive agents in economic theory. Am Econ Rev 81:365–370
Huang C-L, Chen M-C, Wang C-J (2007) Credit scoring with a data mining approach based on support vector machines. Expert Syst Appl 33(4): 847–856
Huang Z, Chen H, Hsu C-J, Chen W-H, Wu S (2004) Credit rating analysis with support vector machines and neural networks: a market comparative study. Decis Support Syst 37(4):543–558
Hull J (2008) Options, futures and other derivatives. Prentice Hall series in finance. Prentice Hall, Upper Saddle River
Huo Y, Cai Z, Gong W, Liu Q (2014) A new adaptive Kalman filter by combining evolutionary algorithm and fuzzy inference system. In: 2014 IEEE congress on evolutionary computation (CEC), Beijing, pp 2893–2900
Jazwinski AH (1970) Stochastic processes and filtering theory. Academic Press, New York
Julier SJ, Uhlmann JK (1997) A new extension of the Kalman filter to nonlinear systems. In: International symposium on aerospace/defense sensing, simulation and controls, Orlando, vol 3, pp 182–193
Kalman RE (1960) A new approach to linear filtering and prediction problems. J Fluids Eng 82(1):35–45
Kirman AP (1992) Whom or what does the representative individual represents? J Econ Perspect 6: 117–136
Kumar KS, Dustakar NR, Jatoth RK (2010) Evolutionary computational tools aided extended Kalman filter for ballistic target tracking. In: 2010 3rd international conference on emerging trends in engineering and technology (ICETET), Goa, pp 588–593
Kumar S, Thulasiram RK, Thulasiraman P (2008) A bioinspired algorithm to price options. In: C3S2E'08: proceedings of the 2008 C3S2E conference. ACM, New York, pp 11–22
LeBaron B (2001) A builder's guide to agent based financial markets. Quant Financ 1:254–261
LeBaron B (2006) Agent-based computational finance. In: Judd KL, Tesfatsion L (eds) Handbook of computational economics, volume 2: agent-based computational economics. Handbooks in economics, chapter 24. North-Holland, pp 1187–1234
Levy M, Levy H, Solomon S (1994) A microscopic model of the stock market: cycles, booms and crashes. Econ Lett 45:103–111
Lipinski P (2004) Evolutionary data-mining methods in discovering stock market expertise from financial time series. PhD thesis, University of Wroclaw, Wroclaw
Lux T (1998) The socio-economic dynamics of speculative markets: interacting agents, chaos, and the fat tails of return distributions. J Econ Behav Organ 33:143–165
Lux T, Ausloos M (2002) Market fluctuations I: scaling, multiscaling and their possible origins. In: Bunde A, Kropp J, Schellnhuber HJ (eds) Theories of disaster – scaling laws governing weather, body, and stock market dynamics. Springer, Berlin Heidelberg pp 373–409
Maringer D (2005) Portfolio management with heuristic optimization. Volume 8 of advances in computational management science. Springer Dordrecht, The Netherlands
Maringer D (2006) Small is beautiful: diversification with a limited number of assets. Working paper WP005-06, Centre for Computational Finance and Economic Agents, University of Essex
Martens D, Baesens B, Gestel TV, Vanthienen J (2007) Comprehensible credit scoring models using rule extraction from support vector machines. Eur J Oper Res 183(3):1466–1476
Martinez-Jaramillo S, Tsang EPK (2009a) Evolutionary computation and artificial financial markets. In: Natural computing in computational finance. Volume 185 of studies in computational intelligence. Springer, Berlin/Heidelberg, pp 137–179
Martinez-Jaramillo S, Tsang EPK (2009b) An heterogeneous, endogenous and coevolutionary gp-based financial market. IEEE Trans Evol Comput 13:33–55
Ong C-S, Huang J-J, Tzeng G-H (2005) Building credit scoring models using genetic programming. Expert Syst Appl 29(1):41–47
O'Sullivan C (2007) Parameter uncertainty in Kalman filter estimation of the cir term structure model. Centre for Financial Markets working paper series

WP-07-18, Centre for Financial Markets, School of Business, University College Dublin
Paleologo G, Elisseeff A, Antonini G (2010) Subagging for credit scoring models. Eur J Oper Res. 201(2):490–499
Potvin J-Y, Soriano P, Vallée M (2004) Generating trading rules on the stock markets with genetic programming. Comput Oper Res 31(7):1033–1047
Quintana D, Saez Y, Mochon A, Isasi P (2008) Early bankruptcy prediction using enpc. Appl Intell 29(2):157–161
Rezaei N, Kordabadi H, Elkamel A, Jahanmiri A (2008) An optimal extended Kalman filter designed by genetic algorithms. Chem Eng Commun 196(5):602–615
Ross SA (1976) The arbitrage theory of capital asset pricing. J Econ Theory 13(3):341–360
Schoreels C, Logan B, Garibaldi JM (2004) Agent based genetic algorithm employing financial technical analysis for making trading decisions using historical equity market data. In: IAT'04: proceedings of the intelligent agent technology, IEEE/WIC/ACM international conference, Washington, DC. IEEE Computer Society, pp 421–424
Sharpe WF Capital asset prices: a theory of market equilibrium under conditions of risk*. J Financ 19(3):425–442 (1964)
Shin K-S, Lee Y-J (2002) A genetic algorithm application in bankruptcy prediction modeling. Expert Syst Appl 23(3):321–328
Simon HA (1982) Models of bounded rationality, vol 2. MIT Press, Cambridge, MA
Streichert F, Ulmer H, Zell A (2004) Evaluating a hybrid encoding and three crossover operators on the constrained portfolio selection problem. In: Proceedings of the 2004 congress on evolutionary computation. IEEE Press, pp 932–939
Sung TK, Chang N, Lee G (1999) Dynamics of modeling in data mining: interpretive approach to bankruptcy prediction. J Manag Inf Syst 16(1): 63–85
Tesfatsion L (2002) Agent-based computational economics: growing economies from the bottom up. Artif Life 8:55–82
Tsang EPK, Martinez-Jaramillo S (2004) Computational finance. In: IEEE computational intelligence society newsletter. 3(8):8–13
Tsang EPK, Yung P, Li J (2004) Eddie-automation, a decision support tool for financial forecasting. J Decis Support Syst Spec Issue Data Min Financ Decis Mak 37(4):559–565
Turku BB, Back B, Laitinen T, Sere K, Wezel MV (1996) Choosing bankruptcy predictors using discriminant analysis, logit analysis, and genetic algorithms. In: Proceedings of the first international meeting on artificial intelligence in accounting, finance and tax, p 337356
West D (2000) Neural network credit scoring models. Comput Oper Res 27(11–12):1131–1152
Yang J (2002) The efficiency of an artificial double auction stock market with neural learning agents. In Evol Comput Econ Financ 85–106, Physica-Verlag Heidelbergh New York
Yin Z, Brabazon A, O'Sullivan C (2007) Adaptive genetic programming for option pricing. In: GECCO'07: proceedings of the 2007 GECCO conference companion on genetic and evolutionary computation. ACM, New York, pp 2588–2594
Zhou Q, Lin C, Yang W (2006) Multi-classifier combination for banks credit risk assessment. In: 1st IEEE conference on industrial electronics and applications, pp 1–4

Evolutionary Computational Techniques in Marketing

Alma Lilia García-Almanza[1], Biliana Alexandrova-Kabadjova[2], and Serafín Martínez-Jaramillo[3]
[1]Directorate of Regulation and Supervision, Banco de México, Mexico City, Mexico
[2]Banco de México, Mexico City, Mexico
[3]Directorate of Financial System Risk Analysis, Banco de México, Mexico City, Mexico

Motivation and Background

The Internet and social networks are key factors that have strongly affected market competition, as they provide customers with more choice in products, services, and prices. For instance, well-established electronic commerce companies such as Amazon, Booking, TripAdvisor, and others provide rankings of their products based on past customer reviews. In the same vein, social networks are a powerful tool to spread good or bad comments about products or services, and they can directly influence potential clients, since the members of the same social network usually share interests and have similar economic levels. For those reasons, marketing teams have focused efforts on creating intelligent business strategies. New artificial intelligence approaches to marketing have emerged, especially evolutionary algorithms used to solve a variety of marketing problems such as the design of attractive products and services for consumers, the analysis of

populations or social networks to target potential clients, the design of new marketing strategies, and more. Nowadays, a huge amount of data on almost any kind of human activity has been stored in structured and unstructured forms. The data is a gold mine, the analysis of which can provide useful information for competing more efficiently in the market. For that reason, machine learning techniques have been used to discover useful patterns for creating user-friendly interfaces and new market segments, among other aims. Many evolutionary computational techniques have been applied to marketing problems in order to obtain a commercial advantage over competitors.

Applications

Marketing is a very dynamic area, as it evolves alongside technology and aims to keep promoted products alive in the market.

The Design of New Products

One of the goals of marketing is to discover products of superior value and quality. To achieve this goal in Fruchter et al. (2006), the authors propose to design a product line rather than a single product. The authors argue that by offering a product line, the manufacturer can customize products according to the needs of different market niches, which would result in higher customer satisfaction and more buyers. Nevertheless, as the time required by the amount of data on customer preferences increases, the optimization process of the product line becomes very hard to manage. For that reason, the authors applied the use of genetic algorithms (GAs) to solve the problem heuristically, and the performance of each solution was valued according to the manufacturer's profits. In a similar way, Liu and Ong (2008) used a GA to solve a marketing segmentation problem. In this case, the evolutionary algorithm was applied to reach all customers effectively.

In the approach proposed by Sundar Balakrishnan and Jacob (1996), a GA was used to optimize for customer preference in product design. The authors followed a three-step methodology in order to create a new product. First, the set of attributes subject to adjustment, such as color or shape, was determined. Second, customer preferences were collected. Finally, a GA was applied to select those attributes that satisfy a larger number of customers.

Targeting Potential Clients

Bhattacharyya (2000) proposed a GA in combination with a case-based reasoning (CBR) system to predict customer purchasing behavior. The objective was to identify potential customers for a specific product or service. This approach was developed and tested with real cases by direct marketing from a worldwide insurance company. An optimization mechanism was integrated into the classification system in order to select those customers most likely to acquire an insurance.

Advertisement

Advertisement is an important area of marketing. It is defined as the activity of attracting public attention to a product or business. Since personalized advertisement improves marketing efficiency, Kwon and Moon (2001) proposed a personalized prediction model to be used in email marketing. A circuit model combined with genetic programs (GPs) was proposed to analyze customer information. The result was a set of recommended rules. It was tested over a general mass marketing. According to the authors, the model showed a significant improvement in sales. In another approach, Naik et al. (1998) used a GA combined with a Kalman filter procedure to determine the best media schedule for advertisement, which at the time was constrained by a budget. This approach evaluated a large number of alternative media schedules to decide upon an optimal media planning solution. The Internet has become very popular and convenient for offering and purchasing, since many products and services can be found easily in a very short time, increasing competition among providers. Since these kinds of sales do not directly involve human interaction, it is essential to design new and better strategies to personalize Web pages. For instance, Abraham and Ramos (2003) proposed an ant clustering algorithm to discover Web usage pat-

terns and a linear genetic programming to analyze visitor behavior. The objective was to discover useful knowledge from user interactions with the Web. The knowledge was used to design adaptive Web sites, business and support services, personalization, network traffic flow analysis, and more.

According to Scanlon (2008), the company Staples used a software called Affinnova to redesign and relaunch its paper brand. Affinnova was designed by Waltham, and it uses a GA to simulate the evolution of consumer markets where strong products survive and weak ones die out. The strongest possible design emerges after several generations. A panel of 750 consumers selected their favorite options from each generation. The software analyzed customer choices over multiple generations to identify preference patterns. Surveys included consumer profiles that contain basic demographic information, customer beliefs, and consumer habits. Clients can also segment results and understand how different designs appeal to different consumers. Affinnova's research also helped to identify the imagery and messaging that would most appeal to consumers.

To summarize, EC has been used to solve a wide variety of marketing problems. Given that ECs are global optimization methods, they can be applied to forecasting and data mining. In this respect, they have great potential for use in the field of marketing. EC techniques allow for the extraction and analysis of customer patterns among large amounts of data, and forecasts of purchasing tendencies, among many other aims.

Cross-References

- ▸ Evolutionary Algorithms
- ▸ Evolutionary Computation in Economics
- ▸ Evolutionary Computation in Finance
- ▸ Genetic and Evolutionary Algorithms
- ▸ Genetic Programming

Recommended Reading

Abraham A, Ramos V (2003) Web usage mining using artificial ant colony clustering and linear genetic programming. In: Congress on evolutionary computation (CEC), Canberra, vol 2. IEEE, pp 1384–1391

Bhattacharyya S (2000) Evolutionary algorithms in data mining: multi-objective performance modeling for direct marketing. In KDD'00: proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining, New York. ACM, pp 465–473

Fruchter G, Fligler A, Winer R (2006) Optimal product line design: a genetic algorithm approach to mitigate cannibalization. J Optim Theory Appl 131(2):227–244

Kwon Y-K, Moon B-R (2001) Personalized email marketing with a genetic programming circuit model. In: Spector L, Goodman ED, Wu A, Langdon WB, Voigt H-M, Gen M, Sen S, Dorigo M, Pezeshk S, Garzon MH, Burke E (eds) Proceedings of the genetic and evolutionary computation conference (GECCO-2001), San Francisco. Morgan Kaufmann, pp 1352–1358

Liu H-H, Ong C-S (2008) Variable selection in clustering for marketing segmentation using genetic algorithms. Expert Syst Appl 34(1):502–510

Naik PA, Mantrala MK, Sawyer AG (1998) Planning media schedules in the presence of dynamic advertising quality. Mark Sci 17(3):214–235

Scanlon J, (2008) "Staples' Evolution", Bloomberg.com, Bloomberg, <http://www.bloomberg.com/news/articles/2008-12-29/staples-evolutionbusinessweek-business-news-stock-market-and-financial-advice>.

Sundar Balakrishnan PV, Jacob VS (1996) Genetic algorithms for product design. Manag Sci 42(8):1105–1117

Evolutionary Computing

▸ Evolutionary Algorithms

Evolutionary Constructive Induction

▸ Evolutionary Feature Selection and Construction

Evolutionary Feature Selection

▸ Evolutionary Feature Selection and Construction

Evolutionary Feature Selection and Construction

Krzysztof Krawiec
Poznan University of Technology, Poznan, Poland

Abstract

Representation of input data has an essential influence on the performance of machine learning systems. Evolutionary algorithms can be used to transform data representation by selecting some of the existing features (evolutionary feature selection) or constructing new features from the existing ones (evolutionary feature construction). This entry provides the rationale for both these approaches and systematizes the research and applications in this area.

Synonyms

EFSC; Evolutionary constructive induction; Evolutionary feature selection; Evolutionary feature synthesis; Genetic attribute construction; Genetic feature selection

Definition

Evolutionary feature selection and construction (EFSC) is a bio-inspired methodology for explicit modification of input data of a learning system. It uses evolutionary computation (EC) to construct a mapping from the original data representation space onto a *secondary* representation space. In evolutionary feature selection (EFS), that mapping consists in dropping off some of the features (▸ attributes) from the original representation so that the dimensionality of the resulting representation space is not greater than that of the original space. In evolutionary feature construction (EFC), an evolutionary algorithm creates (synthesizes) new features (derived attributes) that complement and/or replace the original ones. Therefore, EFS may be considered as a special case of EFC.

A typical EFSC algorithm maintains a population of solutions, each of them encoding a specific mapping. The best mapping found in evolutionary search becomes the data preprocessor for the classifier. Usually, EFSC takes place in the training phase only, and the evolved mapping does not undergo further changes in the testing phase.

Though EFSC is technically a form of data preprocessing (see ▸ Data Preparation), some of its variants may as well involve an internal inductive process in the fitness function. Also, EFS and EFC may be considered as special cases of ▸ Feature Selection and ▸ Feature Construction, respectively. EFC is also partially inspired by ▸ Constructive Induction.

Motivation and Background

Real-world machine-learning problems often involve a multitude of attributes, which individually have low informative content and cannot provide satisfactory performance of the learning system. This applies in particular to data-abundant domains like image analysis and signal processing. When faced with many low-quality attributes, induction algorithms tend to build classifiers that perform poorly in terms of classification accuracy. This problem may be alleviated by removing some features from the original representation space (*feature selection*) or introducing new features defined as informative expressions (arithmetic, logical, etc.) built of *multiple* attributes (*feature construction*).

Many learning algorithms lack the ability of discovering intricate dependencies between attributes, which is a necessary precondition for successful feature selection and construction. This gap is filled out by EFSC, which uses EC to get rid of superfluous attributes and to construct new features. Benefits of EFSC are similar to those of general ▸ Feature Selection and ▸ Feature Construction and include reduced dimensionality of the input space, better predictive accuracy of the learning system, faster

training and querying, and better readability of the acquired knowledge.

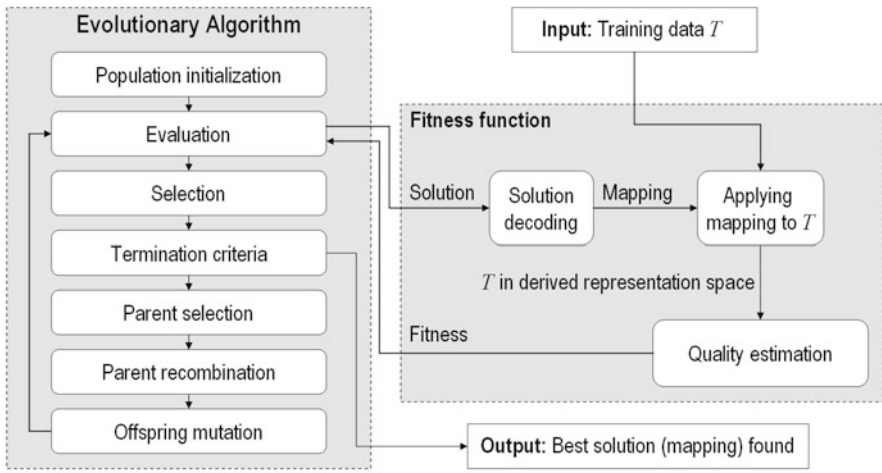
Feature selection and feature construction may be conveniently formulated as an optimization problem with each solution corresponding to a particular feature subset (for feature selection) or to a particular definition of new features (for feature construction). The number of such solutions grows exponentially with the number of original features, rendering the exact search methods infeasible. EC techniques are particularly well-suited to heuristically search these solution spaces. They do not make any assumptions about the optimized function (in contrast to, e.g., the branch-and-bound algorithm) and perform global heuristic search, typically finding high-quality solutions in acceptable time. These virtues are important in EFSC, where the objective function depends on the training data, and it is difficult to predict its properties.

Another strength of EC is easy tailoring to a given task. For instance, a subset of features in EFS is usually encoded as a bit-string solution in genetic algorithm (GA), where a bit at a particular position determines the selection or exclusion of the corresponding feature (Vafaie and Imam 1994; Yang and Honavar 1998). In EFC, definitions of constructed features may be conveniently represented as genetic programming (GP) expressions (Rizki et al. 2002; Teller and Veloso 1997). Also, an evolutionary algorithm naturally produces *many* solutions. This makes it a convenient tool for, e.g., parallel construction of multiple representations (feature subsets) that may be subsequently used in a compound classifier.

Structure of Learning System

Typically, EFSC uses a variant of evolutionary algorithm (usually GA for EFS or genetic programming for EFC) to maintain a population of solutions (individuals), each of them encoding a particular subset of features (for EFS) or definition of new features (for EFC). Solutions undergo mutations, recombinations, and selection. Selective pressure is exerted by a fitness function that estimates a solution's quality by analyzing selected properties of the secondary representation space (see Fig. 1). This usually involves three steps:

1. *Decoding* of solution (retrieving the mapping from the encoded solution).
2. *Transforming* the training set into the secondary representation space according to the mapping.
3. Estimating the *quality* of the secondary representation space, which becomes a solution's fitness.



Evolutionary Feature Selection and Construction, Fig. 1 Evolutionary feature selection and construction

The quality measures employed in step 3 may be grouped into two categories. *Filter approach* relies on the measures that characterize the desired properties of training data in the secondary space (e.g., class separability), abstracting from any particular induction algorithm. *Wrapper approach* estimates the predictive ability in the secondary representation space by a *specific* induction algorithm, usually by partitioning the training set into several subsets and performing multiple train-and-test experiments (e.g., cross-validation). The wrapper approach, though computationally more expensive, takes into account the inductive and representational biases of the employed induction algorithm and thanks to that often proves superior in terms of classification accuracy.

The result of a typical EFSC procedure is the best solution found in an evolutionary run, i.e., the most fit representation mapping. This mapping serves as a preprocessor of input data and is subsequently used to induce the final classifier from the training set. The trained classifier together with the preprocessing provided by the mapping is the final outcome of the EFSC-enriched training process and may be used for classification of new examples.

EFS is the simplest variant of EFSC. In this case, a solution encodes the indices of attributes that should remain in the resulting secondary representation. This leads to straightforward encoding characteristic for GA, with each solution being a bit string as long as the number of original attributes. EFS may be thus easily implemented using off-shelf EA software packages. More sophisticated EFS approaches have been also considered, like evolving GP individuals that *rank* or *score* features (Zhang and Rockett 2011).

Evolutionary feature weighting (EFW) is a direct generalization of EFS, where the evolutionary search weighs the features instead of selecting them. Solutions in EFW are real-valued vectors. EFW requires a wrapper fitness function that can take attribute weights into account. In Komosiński and Krawiec (2000), EFW has been used with a nearest neighbor-based wrapper fitness function to weigh features for a medical diagnosing problem.

EFC usually employs genetic programming to represent feature transformation. Each GP solution encodes an expression tree that uses the original attributes and numeric constants as leaves (terminals) and functions from a predefined vocabulary as internal tree nodes (nonterminals). The value returned by such an expression when applied to an example is interpreted as the new feature. Function set usually encompasses simple arithmetics and elementary functions. The evolved features replace or extend the original ones. If a single new feature is insufficient to provide satisfactory discriminative ability, several GP trees can be encoded within each solution.

EFC is particularly useful in image analysis and computer vision tasks, which naturally tend to involve large numbers of attributes. In such contexts, an EFC algorithm evolves GP solutions that construct higher-level features from low-level image attributes (Krawiec and Bhanu 2005) or implement advanced feature detectors (Howard et al. 2006; Puente et al. 2009). Alternatively, solutions encode chains of operations that process the entire image globally according to the goal specified by the fitness function. Other representations of EFC solutions have been studied as well in GP, including, e.g., graphs (Teller and Veloso 1997) or sequences of operations (Bhanu et al. 2005).

It has been demonstrated that an EFC task may be decomposed into several semi-independent subtasks using cooperative coevolution, a variant of evolutionary algorithm that maintains several populations hosting individuals that encode partial solutions (Krawiec and Bhanu 2005). Other work demonstrates that fragments of GP expressions encoding feature definitions may help to discover good features in other learning tasks (Jaśkowski, Krawiec, and Wieloch 2007).

Applications

Real-world applications of EFSC are numerous and include medical and technical diagnosing,

genetics, detection of intrusions in computer networks, air quality forecasting, brain-computer interfaces, seismography, robotics, face recognition, handwriting recognition, vehicle detection in visual, infrared, and radar modality, image segmentation, satellite imaging, and stereovision. EFS has been built into several machine learning and neural network software packages (e.g., WEKA, Statistica). A ready-to-use implementation of EFC is available in RapidMiner; alternatively, it can be facilitated with the existing EC frameworks like ECJ (http://cs.gmu.edu/~eclab/projects/ecj/). More examples of real-world applications of EFSC may be found in Langdon et al. (2009).

Future Directions

Nowadays, EFC becomes more and more unified with GP-based classification and regression, where solutions are expected to perform the complete classification or regression task rather than to implement only feature definitions. Recently, EFSC has also witnessed the growing popularity of the *multiobjective* evolutionary techniques. In EFC, it is now common to include the complexity of feature definition (reflected by program size in GP) as an additional objective alongside the accuracy of classification (Neshatian and Zhang 2011). This is intended to reduce the so-called *program bloat* (the excessive growth of programs that often pesters GP systems) and so curtail overfitting, because complex features are less likely to generalize well. Other studies involve more "helper objectives," like Bayes error estimate (Olague and Trujillo 2012) or Fisher criterion. Domain-specific measures are also occasionally employed in this character. For instance, in a computer vision study (Arnaldo et al. 2014), interest point detectors are evolved using three objectives that capture detector's stability, spatial dispersion of detected points, and their information content.

The online genetic programming bibliography (Langdon et al. 2009) covers most of the works in evolutionary feature selection and construction. A concise review of contemporary GP research involving feature construction for image analysis and object detection may be found in Krawiec et al. (2007). A systematization of different evolutionary approaches to feature construction is also presented in Bhanu et al. (2005).

Cross-References

- ▸ Constructive Induction
- ▸ Data Preparation
- ▸ Feature Selection

Recommended Reading

Arnaldo I, Krawiec K, O'Reilly U-M (2014) Multiple regression genetic programming. In: Igel C, Arnold DV, Gagne C, Popovici E, Auger A, Bacardit J, Brockhoff D, Cagnoni S, Deb K, Doerr B, Foster J, Glasmachers T, Hart E, Heywood MI,Iba H, Jacob C, Jansen T, Jin Y, Kessentini M, Knowles JD, Langdon WB, Larranaga P, Luke S, Luque G, McCall JAW, Montes de Oca MA, Motsinger-Reif A, Ong YS, Palmer M, Parsopoulos KE, Raidl G, Risi S, Ruhe G, Schaul T, Schmickl T, Sendhoff B, Stanley KO, Stuetzle T, Thierens D, Togelius J, Witt C, Zarges C (eds) GECCO '14: proceedings of the 2014 conference on genetic and evolutionary computation, SIGEVO, Vancouver, 12–16 July. ACM, New York, pp 879–886. doi:10.1145/2576768.2598291, ISBN 978-1-4503-2662-9, http://doi.acm.org/10.1145/2576768.2598291

Bhanu B, Lin Y, Krawiec K (2005) Evolutionary synthesis of pattern recognition systems. Springer, New York

Howard D, Roberts SC, Ryan C (2006) Pragmatic genetic programming strategy for the problem of vehicle detection in airborne reconnaissance. Pattern Recognit Lett 27(11):1275–1288

Jaśkowski W, Krawiec K, Wieloch B (2007) Knowledge reuse in genetic programming applied to visual learning. In: Thierens D et al (eds) GECCO'07: proceedings of the 9th annual conference on genetic and evolutionary computation, vol 2. ACM Press, London, pp 1790–1797

Komosiński M, Krawiec K (2000) Evolutionary weighting of image features for diagnosing of CNS tumors. Artif Intell Med 19(1):25–38

Krawiec K, Bhanu B (2005) Visual learning by coevolutionary feature synthesis. IEEE Trans Syst Man Cybern Part B 35(3):409–425

Krawiec K, Howard D, Zhang M (2007) Overview of object detection and image analysis by means of

genetic programming techniques. In: Proceedings of frontiers in the convergence of bioscience and information technologies 2007 (fbit2007), Jeju, 11–13 oct 2007. IEEE CS Press, pp 779–784

Langdon W, Gustafson S, Koza J (2009) The genetic programming bibliography. http://www.cs.bham.ac.uk/~wbl/biblio/ [online]

Neshatian K, Zhang M (2011) Using genetic programming for context-sensitive feature scoring in classification problems. Connect Sci 23(3):183–207. doi:10.1080/09540091.2011.630065, http://www.tandfonline.com/doi/abs/10.1080/09540091.2011.630065, http://www.tandfonline.com/doi/pdf/10.1080/09540091.2011.630065

Olague G, Trujillo L (2012) Interest point detection through multiobjective genetic programming. Appl Soft Comput 12(8):2566–2582. doi:10.1016/j.asoc.2012.03.058, ISSN 1568-4946, http://www.sciencedirect.com/science/article/pii/S1568494612001706

Puente C, Olague G, Smith SV, Bullock SH, González-Botello MA, Hinojosa-Corona A (2009) A novel GP approach to synthesize vegetation indices for soil eros ion assessment. In: Giacobini M et al (eds) Applications of evolutionary computing. Springer, Berlin/New York, pp 375–384

Rizki MM, Zmuda MA, Tamburino LA (2002) Evolving pattern recognition systems. IEEE Trans Evolut Comput 6(6):594–609

Teller A, Veloso M (1997) PADO: a new learning architecture for object recognition. In: Ikeuchi K, Veloso M (eds) Symbolic visual learning. Oxford Press, New York, pp 77–112

Vafaie H, Imam IF (1994) Feature selection methods: genetic algorithms vs. greedy-like search. In: Proceedings of international conference on fuzzy and intelligent control systems, Louisville, Mar 1994

Yang J, Honavar V (1998) Feature subset selection using a genetic algorithm. IEEE Trans Intell Syst 13(2):44–49

Zhang Y, Rockett PI (2011) A generic optimising feature extraction method using multiobjective genetic programming. Appl Soft Comput 11(1):1087–1097. doi:10.1016/j.asoc.2010.02.008, ISSN 1568-4946, http://www.sciencedirect.com/science/article/B6W86-4YGHGKT-2/2/3c6f14d2e029af14747957a5a2ccfd11

Evolutionary Feature Synthesis

▸ Evolutionary Feature Selection and Construction

Evolutionary Fuzzy Systems

Carlos Kavka
University of Trieste, Trieste, Italy

Definition

An evolutionary fuzzy system is a hybrid automatic learning approximation that integrates ▸ fuzzy systems with ▸ evolutionary algorithms, with the objective of combining the optimization and learning abilities of evolutionary algorithms together with the capabilities of fuzzy systems to deal with approximate knowledge. Evolutionary fuzzy systems allow the optimization of the knowledge provided by the expert in terms of linguistic variables and fuzzy rules, the generation of some of the components of fuzzy systems based on the partial information provided by the expert, and in some cases even the generation of fuzzy systems without expert information. Since many evolutionary fuzzy systems are based on the use of genetic algorithms, they are also known as *genetic fuzzy systems*. However, many models presented in the scientific literature also use genetic programming, evolutionary programming, or evolution strategies, making the term *evolutionary fuzzy systems* more adequate. Highly related is the concept of *evolutionary neuro-fuzzy systems*, where the main difference is that the representation is based on neural networks. Recently, the related concept of *evolving fuzzy systems* has been introduced, where the main objective is to apply evolutionary techniques to the design of fuzzy systems that are adequate to the control of nonstationary processes, mainly on real-time applications.

Motivation and Background

One of the most interesting properties of a fuzzy system is its ability to represent expert knowledge by using linguistic terms of everyday common use, allowing the description of uncertainty, vagueness, and imprecision in the expert knowl-

edge. The linguistic terms, which are imprecise by their own nature, are, however, defined very precisely by using fuzzy theory concepts.

The usual approach to build a fuzzy system consists in the definition of the membership functions and the rule base in terms of expert knowledge. Compared with other rule-based approaches, the process of extracting knowledge from experts and representing it formally is simpler, since linguistic terms can be defined to match the terms used by the experts. In this way, rules are defined establishing relations between the input and output variables using these linguistic terms. However, even if there is a clear advantage of using the terms defined as ▸ fuzzy sets, the knowledge extraction process is still difficult and time consuming, usually requiring a very difficult manual fine tuning process. It should be noted that no automatic framework to determine the parameters of the components of the fuzzy system exists yet, generating the need for methods that provide adaptability and learning ability for the design of fuzzy systems.

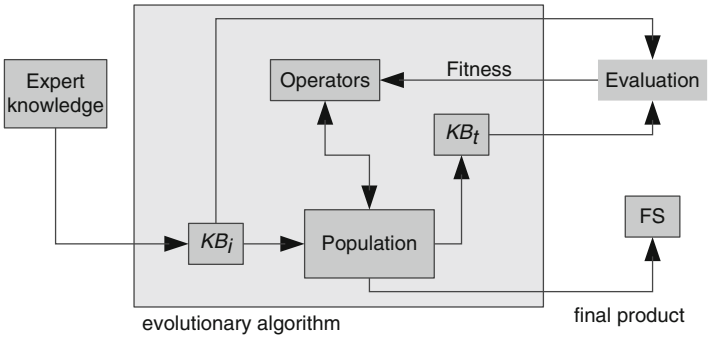
Since it is very easy to map a fuzzy system into a feedforward neural network structure, it is not surprising that many methods based on neural network learning have been proposed to automate the fuzzy system building process (Hoffmann 2001; Karr and Gentry 1993). The combined approach provides advantages from both worlds: the low level learning and computational power of neural networks is joined together with the high level human-like thinking and reasoning of fuzzy systems. However, this approach can still face some problems, such as the potential risk of its learning algorithms to get trapped in local minimum, the possible need for restriction of the membership functions to follow some mathematical properties (like differentiability), and the difficulties of inserting or extracting knowledge in some approaches, where the obtained linguistic terms can exhibit a poor semantic due to the usual black-box processing of many neural networks models.

Evolutionary algorithms provide a set of properties that make them ideal candidates for the optimization and design of fuzzy systems, and in fact, there are many methods that have been proposed in the literature to design or tune the different components of fuzzy systems. Evolutionary systems exhibit robust performance and global search characteristics, while requiring only a simple quality measure from the environment. There is no need for gradient information or input/output patterns. Other strengths come from its parallel nature: instead of selecting a single solution and refining it, in most evolutionary methods, a set of alternative solutions is considered and evolved in parallel.

Structure of the Learning System

The learning process defined by an evolutionary fuzzy system starts from the knowledge provided by the expert, which can include all or just some of the components of the knowledge base of a fuzzy system. The evolutionary algorithm that is behind this learning approach can perform the optimization of all the parameters that are provided by the expert, plus the generation of the missing components of the fuzzy system based on the partial specifications provided by the expert.

The model shown in Fig. 1 presents a general architecture of the learning and optimization process in evolutionary fuzzy systems. An initial knowledge base KB_i is built based on the knowledge provided by the expert. Note that KB_i could be (and usually is) a incompletely specified knowledge base. Based on this initial expert knowledge, the evolutionary algorithm creates a population of individuals, which can represent complete fuzzy systems or just a few components of them. The evaluation of the individuals is performed by creating a temporary knowledge base KB_t, which can also be complete or not. By using the information in KB_t, combined with the initial knowledge base KB_i, the individuals are evaluated by determining the error in the approximation of patterns if there are examples available, computing the reinforcement signal (typical situation in control problems), or in any other way depending on the problem characteristics (Babuska 1998; Cordon et al. 2004). The result



Evolutionary Fuzzy Systems, Fig. 1 The general model of the evolutionary fuzzy systems learning and optimization

of the evaluation is typically a single fitness measure, which provides the necessary information for the selection and the variational operators of the evolutionary algorithm. These operators, which can be standard or defined specifically for the problem, combine and mute the individuals based on the fitness value and their specific parameters. The process is repeated till a predefined criterion is fulfilled, obtaining as a final result the fuzzy system *FS*.

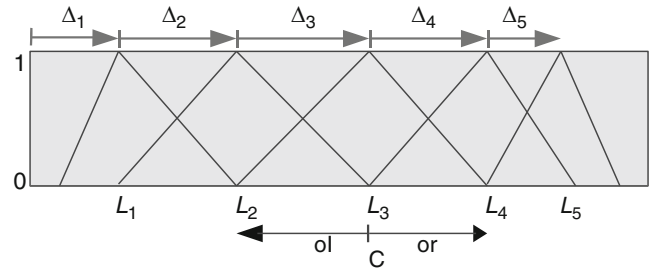
Depending on the information provided by the expert, the learning or optimization process performed by the evolutionary fuzzy system can be applied to the database, the fuzzy rule base or both of them. These three approaches are described below.

Optimization and Learning of the Fuzzy Database

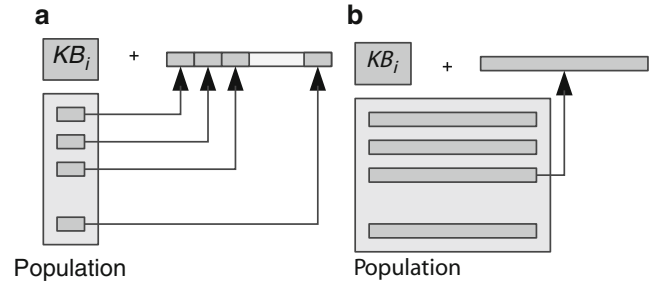
In this case, it is assumed that the fuzzy rule base is known and provided by the expert. The initial knowledge base KB_i contains the fuzzy rule base, and if provided, the initial approximation of the parameters of antecedents and/or consequents. Since the expert has to define the rule base, and in order to do that, he/she needs to know the labels of the linguistic terms used for the antecedents and consequents, it is usual that the number of fuzzy sets is predefined and kept constant during the evolution.

The representation of the individuals contains only the parameters of the fuzzy sets associated to the input linguistic variables, and the fuzzy sets associated to the output variables in the case of a Mamdani fuzzy system, or the associated lineal approximators in the case of a Takagi-Sugeno fuzzy system. Other parameters could also be specified if necessary (scale factors, etc.). Usually, individuals are represented as a fixed length string that is defined as the concatenation of all parameters of the input and output fuzzy sets or approximators. Of course, the representation for the fuzzy sets depends on their particular class: for example, three values are required to represent triangular fuzzy sets, four values to represent trapezoidal fuzzy sets, and two for sigmoidal fuzzy sets. As an example, Fig. 2 shows that three values are necessary to represent a triangular fuzzy set: the center, the left width, and the right width, labeled as *c*, *ol*, and *od*, respectively. From this example, it can be seen that 15 values are required in order to represent the 5 fuzzy sets associated to this single linguistic variable.

However, it is usual to apply fuzzy logic concepts (Zadeh 1988) to simplify the representation, with the implied reduction in the search space, and also, to en- hance the interpretability (Casillas et al. 2003) of the resulting fuzzy system. As an example, it is desirable that the partition associated to a linguistic variable fulfills the completeness property, which establishes that for each point in the input domain, the summation of the membership values of all membership functions must be equal to 1. It is also desirable that the position of the fuzzy sets remains always the same during the evolution, for example in Fig. 2, it means that it is expected that the fuzzy set L_1 will be always at the left of L_2, L_2 always at the left of L_3, and so on. A representation that considers these two requirements can be defined by representing the whole partition specifying the distance from the center of a fuzzy set to the



Evolutionary Fuzzy Systems, Fig. 2 A linguistic variable represented with five fuzzy sets



Evolutionary Fuzzy Systems, Fig. 3 The evaluation of individuals in the (**a**) Michigan and (**b**) Pittsburgh approaches

center of the next one (Hoffmann 2001). The representation of five fuzzy sets then requires only five values (labeled in the figure as Δ_i), which reduces largely the search space and keeps the order of fuzzy sets, while fulfilling the completeness property. Most implementations use real values to represent the parameters.

The operators of the evolutionary algorithm can be standard operators or can be defined specifically based on the selected representation. As an example, operators that modify the width of fuzzy sets, shift the centers, or perform other operations on the fuzzy set representations, linear approximators, or other parameters have been defined in the scientific literature.

Optimization and Learning of the Fuzzy Rule Base

In this case, the fuzzy rule base is not known, or only an initial approximation to it is provided. The other parameters of the knowledge base are known and provided by the expert. The three most usual approximations are

1. Michigan approximation: Each individual of the population codifies a single rule (Bonarini 1996), which means that each individual by itself cannot represent a complete solution to the problem. The knowledge base for evaluation KB_t is built based on the information defined in KB_i and the rules defined by all the individuals from the population combined together (see Fig. 3a). Rules are penalized or rewarded based on its performance during the evaluation. The fuzzy system is then built through the competition of a set of independent rules that have to be learned to collaborate during the evolution.
2. Pittsburgh approximation: Each individual represents the complete rule base. If dynamic creation and removal of rules is allowed, it is necessary to define special variational operators to deal with variable length individuals. Compared with the Michigan approach the evaluation is simpler, since by just combining each individual with KB_i it is possible to build KB_t for evaluation (see Fig. 3b). However, usually, the search space is larger when compared with the Michigan approach.
3. Iterative approximation: Each individual codifies a single rule (Cordon et al. 2001) like in the Michigan approach. However, in each iteration of the algorithm, only the best rule is selected discarding all the others. This selection is based by considering the properties of the rule, such as for example, its covering

degree on a set of examples. The algorithm is then competitive and not cooperative. It is usually necessary to apply algorithms to refine the fuzzy rule set obtained at the end of the evolutionary process, which can include operations, such as for example, the removal of similar rules.

The representation in all of these approximations usually consists of individuals that contain references to the fuzzy sets already defined in KB_i. The representation of each individual can be a sequence of integers where each one is an index to the fuzzy sets associated to the corresponding linguistic variable. As an example, the fuzzy rule base could be represented as a matrix where each cell corresponds to the intersection of the input fuzzy sets, containing the index of the output fuzzy set associated to the rule. It is also possible to represent the fuzzy rule base as a decision table or simply as a list of rules. In these last two cases, the representation can have variable length, allowing to represent fuzzy rule sets with variable size.

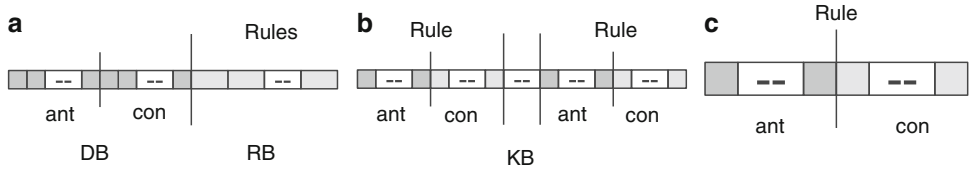
The fitness calculation depends on the selected approximation. On a Pittsburgh approximation, the fitness corresponds to the evaluation of the complete fuzzy system on the corresponding problem. It is also possible to include in the fitness calculation other factors, such as for example, penalization for fuzzy rule bases that contains many rules or fuzzy rules with superposed application areas, etc. On a Michigan or Iterative model, the fitness indicates the degree of adequacy of the rule measured independently, considering also in the Michigan model its degree of cooperation with the other rules in the population.

The definition of the variational operators depends of course on the selected approximation. If the representation allows it, standard operators of crossover and mutation can be used. However, it can be convenient (or necessary) to define specific operators. As an example, variational operators can consider factors such as the time period since the rule has been used for the last time, its overall contribution to the final result, its performance when evaluated on the set of examples, etc.

Optimization and Learning of the Complete Knowledge Base

This case is a combination of the two models described before. The knowledge base KB_i contains the initial approximation to the definition of the antecedents and consequents, and the initial approximation to the fuzzy rule base as provided by the expert. Note that KB_i can also be empty if it is expected that the algorithm must generate all the parameters of the fuzzy system by itself.

The representation of the individuals contains all the parameters that define a knowledge base in order to allow its learning or optimization. The three most used representation schemes are shown in Fig. 4. In the first scheme, each individual contains the representation of all fuzzy sets, and the representation of all fuzzy rules using indexes to refer to the corresponding fuzzy sets. In the second scheme, each individual is structured as a set of rules, where each one specifies its own input and output fuzzy sets by directly including the parameters that define them. The representation (a) is adequate for descriptive fuzzy systems, since the rules contain references



Evolutionary Fuzzy Systems, Fig. 4 Representations for the complete knowledge base adequate for (**a**) descriptive and (**b**) approximative fuzzy systems in the Pittsburgh approximation, and (**c**) representation of a single independent rule adequate for Michigan and Iterative approximations

to the fuzzy sets used in their definition and can be shared by all of them. The representation (b) is adequate for approximative fuzzy systems, where each rule defines its own fuzzy sets. These two representations are adequate for the Pittsburgh approximation, while the third one (c) is adequate for the Michigan and the Iterative approximation. Of course, there can be many variations of this representations. For example, the input space partition can be predefined or obtained through fuzzy clustering algorithms, and if this partition is not expected to go under optimization, then it is not necessary to include the parameters of the input fuzzy sets in the representation.

Since this model is a combination of the two previous models, everything that was mentioned before concerning the fitness function and the variational operators also applies in this context. However, the fact that all parameters of the knowledge base are included in the representation allows to define more powerful variational operators. As an example, it is possible to define operators that decide the creation of new fuzzy sets, the elimination of some of them, and at the same time, the adaptation of the associated fuzzy rules, when for example, it is detected that there are areas in the input space that are not well covered, many rules with superimposed areas, etc. It is also possible to apply genetic programming techniques (Pedrycz 2003), which are usually used to modify the structure of the fuzzy system, adding, removing, or combining sections of the fuzzy system with the objective of generating the most adequate structure.

Final Remarks

Clearly, the integration of fuzzy systems with evolutionary algorithms allows to overcome the limitations of each model considered independently, obtaining a powerful hybrid approach, which allows to learn and optimize fuzzy systems based on expert knowledge. Previous sections have discussed in general terms the evolutionary learning model. However, in order to get more details about particular implementations, it is recommended to read the publications referenced in the next section. The presentation from Karr and Gentry (1993) is interesting, not only because it provides a nice introduction and application of evolutionary fuzzy systems, but it has the additional value of being one of the first publications in the area. The presentation of Hoffmann (2001) is an excellent introduction to evolutionary fuzzy systems used for control applications. The other publications present details on evolutionary fuzzy systems (Babuska 1998; Bonarini 1996; Cordon et al. 2001; Juang et al. 2000; Lee and Takagi 1993), including representations based on neural networks (Hoffmann 2001; Karr and Gentry 1993), evolution strategies (Alpaydtn et al. 2002), genetic programming (Pedrycz 2003) and applications of evolutionary fuzzy systems to the domain of recurrent fuzzy systems (Kavka et al. 2005). The paper by Cordon et al. (2004) provides a very comprehensive reference list about the main developments on evolutionary fuzzy systems.

It should be stressed that a very important aspect to consider in the definition of evolutionary fuzzy systems is the interpretability of the resulting fuzzy systems (Casillas et al. 2003). Even if it has been mentioned that it is possible to design an evolutionary fuzzy system without expert information, by allowing the evolutionary algorithm to define all the components of the knowledge base by itself, it must always be considered that the interpretability of the results is essential. Designing a system that solves the problem, but that works as a black box, can be adequate in other contexts, but it is not desirable at all in the context of evolutionary fuzzy systems. An evolutionary fuzzy system algorithm must provide the means so that the expert knowledge defined in fuzzy terms can be considered and used appropriately during the evolution, and also, it must guarantee an adequate interpretability degree of the resulting fuzzy system.

Recommended Reading

Alpaydtn G, Dundar G, Balktr S (2002) Evolution-based design of neural fuzzy networks using self-adapting genetic parameters. IEEE Trans Fuzzy Syst 10(2):211–221

Babuska R (1998) Fuzzy modeling for control. Kluwer Academic Press, Norwell

Bonarini A (1996) Evolutionary learning of fuzzy rules: competition and cooperation. In: Pedrycz W (ed) Fuzzy modeling: paradigms and practice. Kluwer Academic Press, Norwell
Casillas J, Cordon O, Herrera F, Magdalena L (eds) (2003) Interpretability issues in fuzzy modeling. Studies in fuzziness and soft computing, vol 128. Springer, Berlin/New York
Cordon O, Gomide F, Herrera F, Hoffmann F, Magdalena L (2004) Ten years of genetic fuzzy systems: current framework and new trends. Fuzzy Sets Syst 141:5–31
Cordon O, Herrera F, Hoffmann F (2001) Genetic fuzzy systems. World Scientific Publishing, Singapore
Hoffmann F (2001) Evolutionary algorithms for fuzzy control system design. Proc IEEE 89(9):1318–1333
Juang CF, Lin JY, Lin CT (2000) Genetic reinforcement learning through symbiotic evolution for fuzzy controller design. IEEE Trans Syst Man Cybern 30(2):290–302
Karr CL, Gentry EJ (1993) Fuzzy control of PH using genetic algorithms. IEEE Trans Fuzzy Syst 1(1): 46–53
Kavka C, Roggero P, Schoenauer M (2005) Evolution of Voronoi based fuzzy recurrent controllers. In: Proceedings of GECCO. ACM Press, NeW York, pp 1385–1392
Lee M, Takagi H (1993) Integrating design stages of fuzzy systems using genetic algorithms. In: Proceedings of the second IEEE international conference on fuzzy systems, San Francisco, pp 612–617
Pedrycz W (2003) Evolutionary fuzzy modeling. IEEE Trans Fuzzy Syst 11(5):652–665
Zadeh L (1988) Fuzzy logic. IEEE Comput 21(4): 83–93

Evolutionary Games

Moshe Sipper
Ben-Gurion University, Beer-Sheva, Israel

Definition

Evolutionary algorithms are a family of algorithms inspired by the workings of evolution by natural selection, whose basic structure is to:

1. Produce an initial *population* of individuals, these latter being candidate solutions to the problem at hand.
2. Evaluate the *fitness* of each individual in accordance with the problem whose solution is sought.
3. *While* termination condition not met *do*:
   (a) *Select* fitter individuals for reproduction
   (b) *Recombine* (*crossover*) individuals
   (c) *Mutate* individuals
   (d) *Evaluate* fitness of modified individuals
4. *End while*

Evolutionary games is the application of evolutionary algorithms to the evolution of game-playing strategies for various games, including chess, backgammon, and Robocode.

Motivation and Background

Ever since the dawn of artificial intelligence in the 1950s, games have been part and parcel of this lively field. In 1957, a year after the Dartmouth Conference that marked the official birth of AI, Alex Bernstein designed a program for the IBM 704 that played two amateur games of chess. In 1958, Allen Newell, J.C. Shaw, and Herbert Simon introduced a more sophisticated chess program (beaten in 35 moves by a 10-year-old beginner in its last official game played in 1960). Arthur L. Samuel of IBM spent much of the 1950s working on game-playing AI programs, and by 1961, he had a checkers program that could play at the master's level. In 1961 and 1963, Donald Michie described a simple trial-and-error learning system for learning how to play tic-tac-toe (or Noughts and Crosses) called MENACE (for Matchbox Educable Noughts and Crosses Engine). These are but examples of highly popular games that have been treated by AI researchers since the field's inception.

Why study games? This question was answered by Susan L. Epstein, who wrote:

> There are two principal reasons to continue to do research on games... First, human fascination with game playing is long-standing and pervasive. Anthropologists have cataloged popular games in almost every culture... Games intrigue us because they address important cognitive functions... The second reason to continue game-playing research

> is that some difficult games remain to be won, games that people play very well but computers do not. These games clarify what our current approach lacks. They set challenges for us to meet, and they promise ample rewards (Epstein 1999).

Studying games may thus advance our knowledge in both cognition and artificial intelligence, and, last but not least, games possess a competitive angle which coincides with our human nature, thus motivating both researcher and student alike.

Even more strongly, Laird and van Lent proclaimed that:

> . . . interactive computer games are the killer application for human-level AI. They are the application that will soon need human-level AI, and they can provide the environments for research on the right kinds of problems that lead to the type of the incremental and integrative research needed to achieve human-level AI (Laird and van Lent 2000).

Recently, evolutionary algorithms have proven a powerful tool that can automatically "design" successful game-playing strategies for complex games (Azaria and Sipper 2005a,b; Hauptman and Sipper 2005b, 2007a,b; Shichel et al. 2005; Sipper et al. 2007).

Structure of the Learning System

Genetic Programming

Genetic programming is a subclass of evolutionary algorithms, wherein a *population* of individual programs is evolved, each program comprising *functions* and *terminals*. The functions are usually arithmetic and logic operators that receive a number of arguments as input and compute a result as output; the terminals are zero-argument functions that serve both as constants and as sensors, the latter being a special type of function that queries the domain environment.

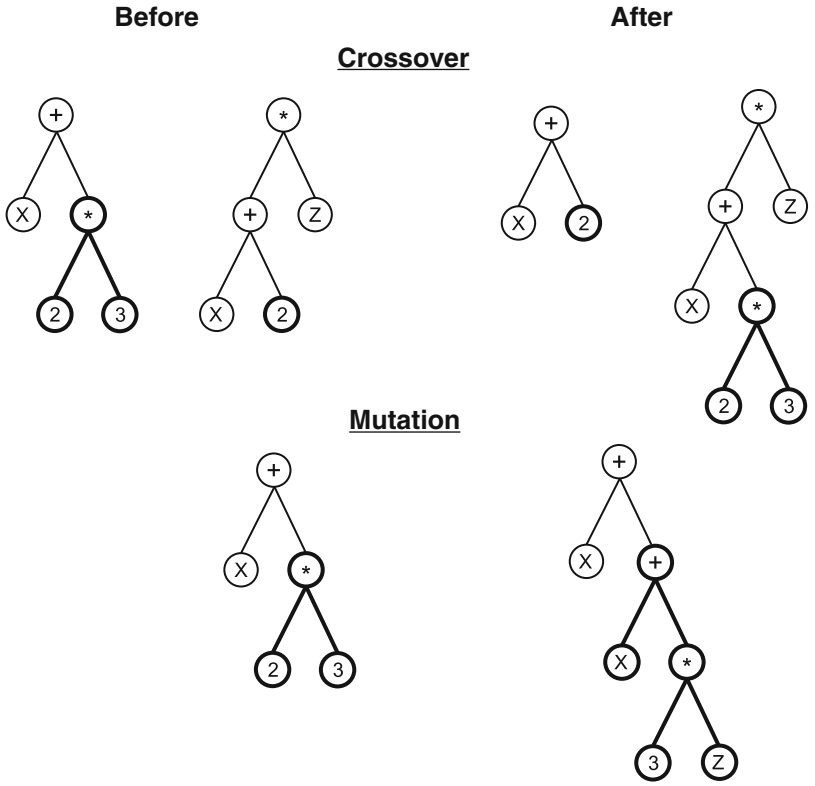
The main mechanism behind genetic programming is precisely that of a generic evolutionary algorithm (Sipper 2002; Tettamanzi and Tomassini 2001), namely, the repeated cycling through four operations applied to the entire population: evaluate-select-crossover-mutate. Starting with an initial population of randomly generated programs, each individual is evaluated in the domain environment and assigned a *fitness* value representing how well the individual solves the problem at hand. Being randomly generated, the first-generation individuals usually exhibit poor performance. However, some individuals are better than others, that is, (as in nature) variability exists, and through the mechanism of natural (or, in our case, artificial) selection, these have a higher probability of being selected to parent the next generation. The size of the population is finite and usually constant.

Specifically, first a genetic operator is chosen at random; then, depending on the operator, one or two individuals are selected from the current population using a *selection operator*, one example of which is *tournament selection*: Randomly choose a small subset of individuals, and then select the one with the best fitness. After the probabilistic selection of better individuals, the chosen genetic operator is used to construct the next generation. The most common operators are:

- Reproduction (unary): Copy one individual to the next generation with no modifications. The main purpose of this operator is to preserve a small number of good individuals.
- Crossover (binary): Randomly select an internal node in each of the two individuals and swap the subtrees rooted at these nodes. An example is shown in Fig. 1.
- Mutation (unary): Randomly select a node from the tree, delete the subtree rooted at that node, and then "grow" a new subtree in its stead. An example is shown in Fig. 1 (the growth operator as well as crossover and mutation are described in detail in Koza 1992).

The generic genetic programming flowchart is shown in Fig. 2. When one wishes to employ genetic programming, one needs to define the following six desiderata:

1. Program architecture
2. Set of terminals



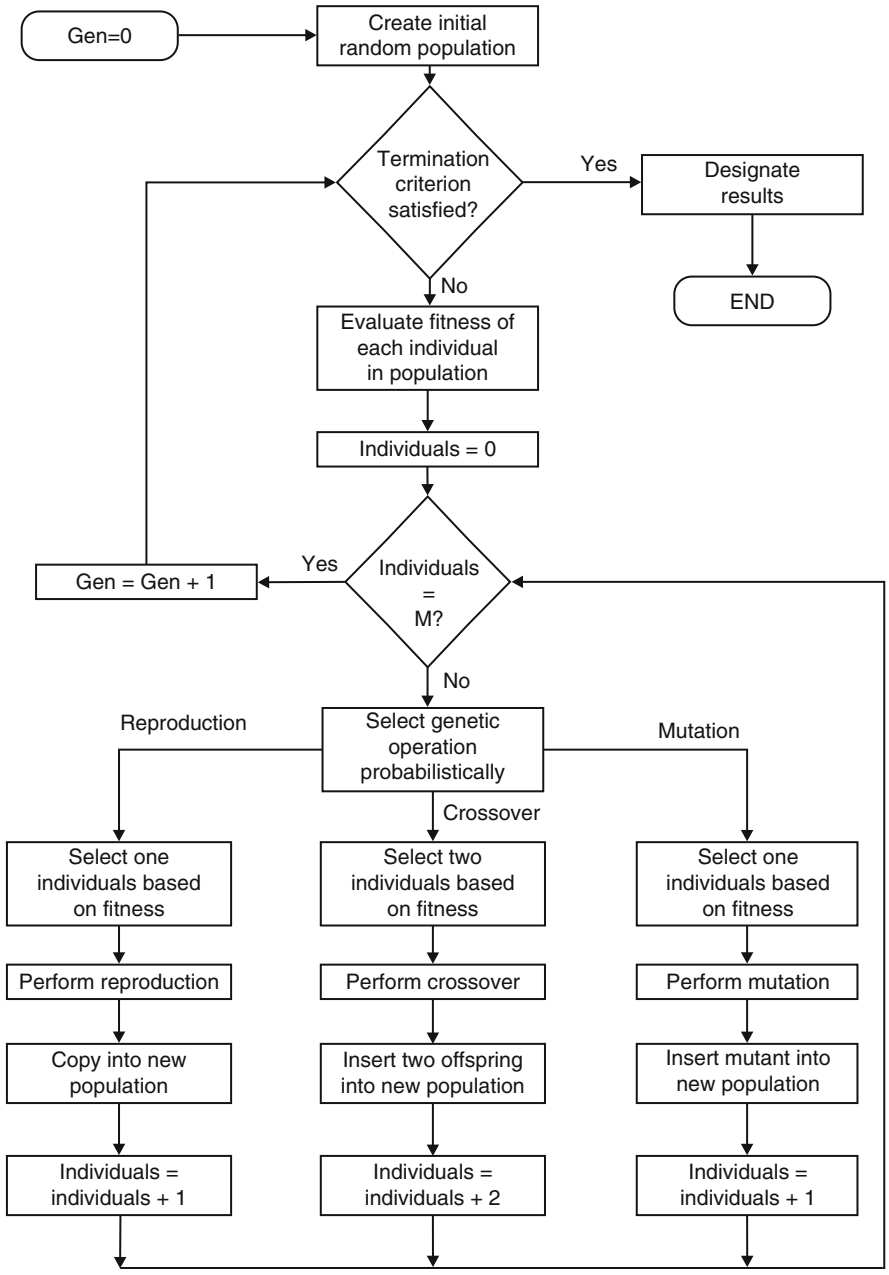
Evolutionary Games, Fig. 1 Genetic operators in genetic programming. LISP programs are depicted as trees. Crossover (*top*): Two subtrees (marked in *bold*) are selected from the parents and swapped. Mutation (*bottom*): A subtree (marked in *bold*) is selected from the parent individual and removed. A new subtree is grown instead

3. Set of functions
4. Fitness measure
5. Control parameters
6. Manner of designating result and terminating run

Evolving Game-Playing Strategies

Recently, we have shown that complex and successful game-playing strategies can be attained via genetic programming. We focused on three games (Azaria and Sipper 2005a,b; Hauptman and Sipper 2005b, 2007a,b; Shichel et al. 2005; Sipper et al. 2007):

1. *Backgammon.* Evolves a full-fledged player for the nondoubling-cube version of the game (Azaria and Sipper 2005a,b; Sipper et al. 2007).
2. *Chess* (endgames). Evolves a player able to play endgames (Hauptman and Sipper 2005b, 2007a,b; Sipper et al. 2007). While endgames typically contain but a few pieces, the problem of evaluation is still hard, as the pieces are usually free to move all over the board, resulting in complex game trees – both deep and with high branching factors. Indeed, in the chess lore, much has been said and written about endgames.
3. *Robocode.* A simulation-based game in which robotic tanks fight to destruction in a closed arena (robocode.alphaworks.ibm.com). The programmers implement their robots in the Java programming language and can test their creations either by using a graphical environment in which battles are held or by submitting them to a central Web site where



Evolutionary Games, Fig. 2 Generic genetic programming flowchart (based on Koza 1992). M is the population size, and Gen is the generation counter. The termination criterion can be the completion of a fixed number of generations or the discovery of a good-enough individual

online tournaments regularly take place. Our goal here has been to evolve Robocode players able to rank high in the international league (Shichel et al. 2005; Sipper et al. 2007).

A strategy for a given player in a game is a way of specifying which choice the player is to make at every point in the game from the set of allowable choices at that point, given all the information that is available to the player at that

point (Koza 1992). The problem of discovering a strategy for playing a game can be viewed as one of seeking a computer program. Depending on the game, the program might take as input the entire history of past moves or just the current state of the game. The desired program then produces the next move as output. For some games, one might evolve a complete strategy that addresses every situation tackled. This proved to work well with Robocode, which is a dynamic game, with relatively few parameters and little need for past history.

In a two-player game, such as chess or backgammon, players move in turn, each trying to win against the opponent according to specific rules (Hong et al. 2001). The course of the game may be modeled using a structure known as an adversarial game tree (or simply game tree), in which nodes are the positions in the game and edges are the moves. By convention, the two players are denoted as MAX and MIN, where MAX is the player who moves first. Thus, all nodes at odd-numbered tree levels are game positions where MAX moves next (labeled MAX nodes). Similarly, nodes on even levels are called MIN nodes and represent positions in which MIN (opponent) moves next.

The complete game tree for a given game is the tree starting at the initial position (the root) and containing all possible moves (edges) from each position. *Terminal nodes* represent positions where the rules of the game determine whether the result is a win, a draw, or a loss. Although the game tree for the initial position is an explicit representation of all possible paths of the game, therefore theoretically containing all the information needed to play perfectly, for most (nontrivial) games, it is extremely large, and constructing it is not feasible. For example, the complete chess game tree consists of roughly 10^{43} nodes (Shannon 1950).

When the game tree is too large to be generated completely, only a partial tree (called a search tree) is generated instead. This is accomplished by invoking a *search algorithm*, deciding which nodes are to be developed at any given time and when to terminate the search (typically at nonterminal nodes due to time constraints). During the search, some nodes are evaluated by means of an *evaluation function* according to given heuristics. This is done mostly at the leaves of the tree. Furthermore, search can start from any position and not just at the beginning of the game.

Because we are searching for a winning strategy, we need to find a good next move for the current player, such that no matter what the opponent does thereafter, the player's chances of winning the game are as high as possible. A well-known method called the *minimax* search (Campbell and Marsland 1983; Kaindl 1988) has traditionally been used, and it forms the basis for most methods still in use today. This algorithm performs a depth-first search (the depth is usually predetermined), applying the evaluation function to the leaves of the tree and propagating these values upward according to the minimax principal: at MAX nodes, select the maximal value and at MIN nodes – the minimal value. The value is ultimately propagated to the position from which the search had started.

With games such as backgammon and chess, one can couple a current-state evaluator (e.g., board evaluator) with a next-move generator. One can then go on to create a minimax tree, which consists of all possible moves, counter moves, counter counter-moves, and so on; for real-life games, such a tree's size quickly becomes prohibitive. The approach we used with backgammon and chess is to derive a very shallow, single-level tree and evolve "smart" evaluation functions. Our artificial player is thus created by combining an evolved board evaluator with a simple program that generates all next-move boards (such programs can easily be written for backgammon and chess).

In what follows, we describe the definition of the six items necessary in order to employ genetic programming, as delineated in the previous section: program architecture, set of terminals, set of functions, fitness measure, control parameters, and manner of designating result and terminating run. Due to lack of space, we shall elaborate upon one game – Robocode – and only summarize the major results for backgammon and chess.

Example: Robocode

Program Architecture

A Robocode player is written as an event-driven Java program. A main loop controls the tank activities, which can be interrupted on various occasions, called *events*. The program is limited to four lines of code, as we were aiming for the HaikuBot category, one of the divisions of the international league with a four-line code limit. The main loop contains one line of code that directs the robot to start turning the gun (and the mounted radar) to the right. This insures that within the first gun cycle, an enemy tank will be spotted by the radar, triggering a *ScannedRobotEvent*. Within the code for this event, three additional lines of code were added, each controlling a single actuator and using a single numerical input that was supplied by a genetic programming-evolved subprogram. The first line instructs the tank to move to a distance specified by the first evolved argument. The second line instructs the tank to turn to an azimuth specified by the second evolved argument. The third line instructs the gun (and radar) to turn to an azimuth specified by the third evolved argument (Fig. 3).

Terminal and Function Sets

We divided the terminals into three groups according to their functionality (Shichel et al. 2005) (Fig. 4):

1. Game-status indicators: A set of terminals that provide real-time information on the game status, such as last enemy azimuth, current tank position, and energy levels.
2. Numerical constants: Two terminals, one providing the constant 0 and the other being an ephemeral random constant (ERC). This latter terminal is initialized to a random real numerical value in the range [−1, 1] and does not change during evolution.
3. Fire command: This special function is used to curtail one line of code by not implementing the fire actuator in a dedicated line.

Fitness Measure

We explored two different modes of learning: using a fixed external opponent as teacher and coevolution – letting the individuals play against each other; the former proved better. However, not just one but three external opponents were used to measure performance; these adversaries were downloaded from the HaikuBot league (robocode.yajags.com). The fitness value of an individual equals its average fractional score (over three battles).

Control Parameters and Run Termination

The major evolutionary parameters (Koza 1992) were population size (256), generation count (between 100 and 200), selection method (tournament), reproduction probability (0), crossover probability (0.95), and mutation probability (0.05). An evolutionary run terminates when fitness is observed to level off. Since the game is highly nondeterministic, a "lucky" individual might attain a higher fitness value than better overall individuals. In order to obtain a more accurate measure for the evolved players, we let each of them do battle for 100 rounds against 12 different adversaries (one at a time). The results were used to extract the

Evolutionary Games, Fig. 3 Robocode player's code layout (HaikuBot division)

Robocode Player's Code Layout

```
while (true)
    TurnGunRight(INFINITY); //main code loop
...
OnScannedRobot(){
    MoveTank(<GP#1>);
    TurnTankRight(<GP#2>);
    TurnGunRight(<GP#3>);
}
```

a

Energy()	Returns the remaining energy of the player
Heading()	Returns the current heading of the player
X()	Returns the current horizontal position of the player
Y()	Returns the current vertical position of the player
MaxX()	Returns the horizontal battlefield dimension
MaxY()	Returns the vertical battlefield dimension
EnemyBearing()	Returns the current enemy bearing, relative to the current player's heading
EnemyDistance()	Returns the current distance to the enemy
EnemyVelocity()	Returns the current enemy's velocity
EnemyHeading()	Returns the current enemy heading, relative to the current player's heading
EnemyEnergy()	Returns the remaining energy of the enemy
Constant()	An ERC (Ephemeral Random Constant) in the range [–1,1]
Random()	Returns a random real number in the range [–1,1]
Zero()	Returns the constant 0

b

Add(F, F)	Add two real numbers
Sub(F, F)	Subtract two real numbers
Mul(F, F)	Multiply two real numbers
Div(F, F)	Divide first argument by second, if denominator non-zero, otherwise return zero
Abs(F)	Absolute value
Neg(F)	Negative value
Sin(F)	Sine function
Cos(F)	Cosine function
ArcSin(F)	Arcsine function
ArcCos(F)	Arccosine function
IfGreater(F, F, F, F)	If first argument greater than second, return value of third argument, else return value of fourth argument
IfPositive(F, F, F)	If first argument is positive, return value of second argument, else return value of third argument
Fire(F)	If argument is positive, execute fire command with argument as fire-power and return 1; otherwise, do nothing and return 0

Evolutionary Games, Fig. 4 Robocode representation. (**a**) Terminal set (**b**) Function set (*F*: Float)

top player – to be submitted to the international league.

Results

We submitted our top player to the HaikuBot division of the international league. At its very first tournament, it came in third, later climbing to first place of 28 (robocode.yajags.com/20050625/haiku-1v1.html). All other 27 programs, defeated by our evolved strategy, were written by humans. For more details on GP-Robocode see Shichel et al. (2005) and Sipper et al. (2007).

Backgammon and Chess: Major Results

Backgammon

We pitted our top evolved backgammon players against *Pubeval*, a free, public-domain board evaluation function written by Tesauro. The program – which plays well – has become the de facto yardstick used by the growing commu-

Evolutionary Games, Table 1 Percent of wins, advantages, and draws for the best GP-EndChess player in the tournament against two top competitors

	%Wins	%Advs	%Draws
Master	6.00	2.00	68.00
CRAFTY	2.00	4.00	72.00

nity of backgammon-playing program developers. Our top evolved player was able to attain a win percentage of 62.4 % in a tournament against Pubeval, about 10 % higher (!) than the previous top method. Moreover, several evolved strategies were able to surpass the 60 % mark, and most of them outdid all previous works. For more details on GP-Gammon, see Azaria and Sipper (2005a) and Sipper et al. (2007).

Chess (Endgames)

We pitted our top evolved chess-endgame players against two very strong external opponents: (1) a program we wrote ("Master") based upon consultation with several high-ranking chess players (the highest being Boris Gutkin, ELO 2400, International Master) and (2) CRAFTY – a world-class chess program, which finished second in the 2004 World Computer Speed Chess Championship (www.cs.biu.ac.il/games/). Speed chess ("blitz") involves a time limit per move, which we imposed both on CRAFTY and on our players. Not only did we thus seek to evolve good players, but ones who play well *and fast*. Results are shown in Table 1. As can be seen, GP-EndChess manages to hold its own, and even wins, against these top players. For more details on GP-EndChess, see Sipper et al. (2007) and Hauptman and Sipper (2005b).

Deeper analysis of the strategies developed (Hauptman and Sipper 2005a) revealed several important shortcomings, most of which stemmed from the fact that they used deep knowledge and little search (typically, they developed only *one* level of the search tree). Simply increasing the search depth would not solve the problem, since the evolved programs examine each board very thoroughly, and scanning many boards would increase time requirements prohibitively. And so we turned to evolution to find an optimal way to overcome this problem: How to add more search at the expense of less knowledgeable (and thus less time-consuming) node evaluators, while attaining better performance. In Hauptman and Sipper (2007b) *we evolved the search algorithm itself*, focusing on the *Mate-In-N* problem: find a key move such that even with the best possible counterplays, the opponent cannot avoid being mated in (or before) move N. We showed that our evolved search algorithms successfully solve several instances of the Mate-In-N problem, for the hardest ones developing 47 % less game-tree nodes than CRAFTY. Improvement is thus not over the basic alpha-beta algorithm, but over a world-class program using all standard enhancements (Hauptman and Sipper 2007b).

Finally, in Hauptman and Sipper (2007a), we examined a strong evolved chess-endgame player, focusing on the player's emergent capabilities and tactics in the context of a chess match. Using a number of methods, we analyzed the evolved player's building blocks and their effect on play level. We concluded that evolution has found combinations of building blocks that are far from trivial and cannot be explained through simple combination – thereby indicating the possible emergence of complex strategies.

Cross-References

- Evolutionary Algorithms
- Evolutionary Computation
- Evolutionary Computing
- Genetic Programming

Recommended Reading

Azaria Y, Sipper M (2005a) GP-Gammon: genetically programming backgammon players. Genet Program Evolvable Mach 6(3):283–300

Azaria Y, Sipper M (2005b) GP-Gammon: using genetic programming to evolve backgammon players.

In: Keijzer M, Tettamanzi A, Collet P, van Hemert J, Tomassini M (eds) Proceedings of 8th European conference on genetic programming (EuroGP2005), Lausanne. LNCS, vol 3447. Springer, Heidelberg, pp 132–142
Campbell MS, Marsland TA (1983) A comparison of minimax tree search algorithms. Artif Intell 20:347–367
Epstein SL (1999) Game playing: the next moves. In: Proceedings of the sixteenth national conference on artificial intelligence, Orland. AAAI, Menlo Park, pp 987–993
Hauptman A, Sipper M (2005a) Analyzing the intelligence of a genetically programmed chess player. In: Late breaking papers at the 2005 genetic and evolutionary computation conference (GECCO 2005), Washington, DC
Hauptman A, Sipper M (2005b) GP-EndChess: using genetic programming to evolve chess endgame players. In: Keijzer M, Tettamanzi A, Collet P, van Hemert J, Tomassini M (eds) Proceedings of 8th European conference on genetic programming (EuroGP2005), Lausanne. LNCS, vol 3447. Springer, Heidelberg, pp 120–131
Hauptman A, Sipper M (2007a) Emergence of complex strategies in the evolution of chess endgame players. Adv Complex Syst 10(Suppl 1):35–59
Hauptman A, Sipper M (2007b) Evolution of an efficient search algorithm for the mate-in-N problem in chess. In: Ebner M, O'Neill M, Ekárt A, Vanneschi L, Esparcia-Alcázar AI (eds) Proceedings of 10th European conference on genetic programming (EuroGP2007), Valencia. LNCS, vol 4445. Springer, Heidelberg, pp 78–89
Hong T-P, Huang K-Y, Lin W-Y (2001) Adversarial search by evolutionary computation. Evol Comput 9(3):371–385
Kaindl H (1988) Minimaxing: theory and practice. AI-Mag 9(3):69–76
Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT, Cambridge
Laird JE, van Lent M (2000) Human-level AI's killer application: interactive computer games. In: AAAI-00: proceedings of the 17th national conference on artificial intelligence, Austin. MIT, Cambridge, pp 1171–1178
Shannon CE (1950) Automatic chess player. Sci Am 48:182
Shichel Y, Ziserman E, Sipper M (2005) GP-Robocode: using genetic programming to evolve robocode players. In: Keijzer M, Tettamanzi A, Collet P, van Hemert J, Tomassini M (eds) Proceedings of 8th European conference on genetic programming (EuroGP2005), Lausanne. LNCS, vol 3447. Springer, Heidelberg, pp 143–154
Sipper M (2002) Machine nature: the coming age of bio-inspired computing. McGraw-Hill, New York
Sipper M, Azaria Y, Hauptman A, Shichel Y (2007) Designing an evolutionary strategizing machine for game playing and beyond. IEEE Trans Syst Man Cybern Part C Appl Rev 37(4):583–593
Tettamanzi A, Tomassini M (2001) Soft computing: integrating evolutionary, neural, and fuzzy systems. Springer, Berlin

Evolutionary Grouping

▸ Evolutionary Clustering

Evolutionary Kernel Learning

Christian Igel
Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

Definition

Evolutionary kernel learning stands for using ▸ evolutionary algorithms to design the ▸ kernel function for a ▸ kernel method.

Motivation and Background

In kernel-based learning algorithms, the kernel function implicitly defines the feature space in which the algorithm operates. The kernel determines the scalar product and thereby the metric in the feature space. Choosing the right kernel function is crucial for the training accuracy and generalization performance of the learning machine. The choice may also influence the runtime and storage complexity during and after training.

The kernel is usually not adapted by the kernel method itself; choosing it is a ▸ model selection problem. In practice, the kernel function is selected from an a priori fixed class. When a parameterized family of kernel functions is considered, kernel adaptation reduces to finding appropriate parameters. The most frequently used method to determine these values is grid search. In simple grid search, the parameters are varied with a fixed step-size through a range of values, and the performance of each combination is measured. Because of its computational complexity,

grid search is only suitable for the adjustment of a few parameters. Furthermore, the choice of the discretization of the search space may be crucial. Gradient-based approaches are perhaps the most elaborate techniques for adapting real-valued kernel parameters, see the articles by Chapelle et al. (2002) and Glasmachers and Igel (2005) and references therein. To use these methods, however, the class of kernel functions must have a differentiable structure. Furthermore, score functions for assessing the parameter performance that are not differentiable and/or piecewise constant may cause problems. Evolutionary kernel learning does not suffer from these limitations. Additionally, it allows for ▸ multi-objective optimization (MOO) to address several kernel design criteria.

Structure of Learning System

Canonical evolutionary kernel learning can be described as an evolutionary algorithm (EA) in which the individuals encode kernel functions, see Fig. 1. These individuals are evaluated by determining the task-specific performance of the kernel they represent. Two special aspects must be considered when designing an EA for kernel learning. First, one must decide how to assess the performance (i.e., the fitness) of a particular kernel. That is, model selection criteria have to be defined depending on the problem at hand. Second, one must also specify the subset of possible kernel functions to be searched by the EA. This leads to the questions of how to encode the kernels and which variation operators to employ.

Assessing Fitness: Model Selection Criteria

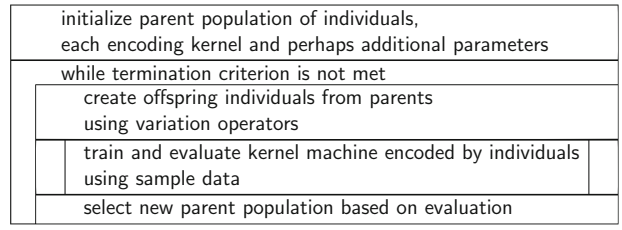
The following presents some performance indices that have been considered for evolutionary kernel learning. They can be used individually or in linear combination for single-objective optimization. In MOO several of these criteria can be used as different objectives.

It is important to note that, although many of these measures are designed to improve ▸ generalization, kernel learning can lead to ▸ overfitting if only limited data are used in the model selection process (e.g., if in every generation, the same small data sets are used to assess performance). Regularization (e.g., in a Bayesian framework) can be used to prevent overfitting. If enough data are available, it is advisable to monitor the generalization behavior of kernel learning using independent data. For example, external data can be used for the early stopping of evolutionary kernel learning (Igel 2013).

Accuracy on Sample Data

The most straightforward way to evaluate a model is to consider its performance on sample data. The empirical risk given by the error on the training data can be considered, but it does not measure generalization. To estimate the generalization performance, the accuracy on data not used for training is evaluated. In the simplest case, the available data are split into a training and validation set, with the first used for learning and the second for subsequent performance assessment. A theoretically sound and simple method is ▸ cross-validation (CV). Cross-validation makes better use of the available data, but it is more computationally demanding.

Using holdout validation sets alone does not prevent overfitting if the validation sets are small and are reused in every generation. If sufficient data are available, it is advisable to resample the data used for fitness evaluation in each generation to prevent overfitting (Igel 2013).



Evolutionary Kernel Learning, Fig. 1
Canonical evolutionary kernel learning algorithm

If ▸ classification is considered, it may be reasonable to split the classification error into false negative and false positive rates and to view ▸ sensitivity and ▸ specificity as two separate objectives (Suttorp and Igel 2006).

Measures Derived from Bounds on the Generalization Performance

Statistical learning theory provides estimates of and bounds on the expected generalization error of learning machines. These results can be utilized as criteria for model selection. One has to keep in mind that the model selection process may lead to violations of the assumptions underlying the corresponding theorems from statistical learning theory, in which case the computed performance indicators can not be strictly interpreted as bounds and unbiased estimates.

An example drawing inspiration from radius-margin bounds for evolving kernels for ▸ support vector machines (SVMs) for classification is given by Igel (2005). Furthermore, the number of support vectors (SVs) was optimized in combination with the empirical risk (Igel 2005). For hard-margin SVMs, the fraction of SVs is an upper bound on the leave-one-out error (e.g., Chapelle et al. 2002).

Number of Input Variables

Variable selection refers to the ▸ feature selection problem of choosing input variables that are best suited for the learning task. Masking a subset of variables can be viewed as modifying the kernel. Considering only a subset of feature dimensions decreases the computational complexity of the learning machine. When deteriorating feature dimensions are removed, the overall performance may increase. Reducing the number of input variables is therefore a common objective, which can be achieved by using single-objective (Eads et al. 2002; Fröhlich et al. 2004; Jong et al. 2004; Miller et al. 2003) or multi-objective (Pang and Kasabov 2004; Shi et al. 2004) evolutionary kernel learning.

Space and Time Complexity of the Classifier

Sometimes it can be very important to have fast kernel methods (e.g., for meeting real-time constraints). Thus, the execution time may be considered in the performance assessment during evolutionary kernel learning.

Reducing the number of input variables speeds up kernel methods. The space and time complexity of SVMs also scales with the number of SVs. This is an additional reason to consider minimization of the number of SVs as an objective in evolutionary model selection for SVMs (Igel 2005; Suttorp and Igel 2006).

Multi-objective Optimization

The design of a learning machine can be considered as a MOO problem. For example, accuracy and complexity can be viewed as different, and probably conflicting, objectives. The goal of MOO is to approximate a diverse set of Pareto-optimal solutions (i.e., solutions that cannot be improved in one objective without getting worse in another one), which provide insights into the trade-offs between the objectives. Evolutionary multi-objective algorithms have become popular for MOO. Applications of multi-objective evolutionary kernel learning combining some of the performance measures listed above can be found in the work of Igel (2005), Pang and Kasabov (2004), Shi et al. (2004), and Suttorp and Igel (2006).

Coevolution

▸ Coevolutionary learning also finds application in evolutionary kernel design. For instance, Gagné et al. (2006) suggest coevolution to speed up the evaluation and optimization of kernel nearest neighbor classifiers. They evolve three different species. The first encodes the kernels, the second a subset of the training examples used for building the classifier, and the third a subset of examples used for fitness evaluation. Kernels and training examples cooperate, while the third species competes with the kernels.

Encoding and Variation Operators

The sheer complexity of the space of possible kernel functions makes it necessary to restrict the search to a particular class of kernel functions. This restriction essentially determines the repre-

sentation and the operators used in evolutionary kernel learning.

When a parameterized family of mappings is considered, the kernel parameters can be encoded more or less directly in a real-valued EA. This is a frequently used representation, for example, for Gaussian kernel functions.

For variable selection, a binary encoding can be appropriate. For choosing a subset out of d variables, bitstrings of length d can be considered, where each bit indicates whether a particular input variable is considered in the kernel computation or not (Pang and Kasabov 2004; Shi et al. 2004).

Kernels can be built from other kernels. For example, if k_1 and k_2 are kernel functions on X, then $ak_1(\boldsymbol{x},\boldsymbol{z}) + bk_2(\boldsymbol{x},\boldsymbol{z})$ and $a\exp(-bk_1(\boldsymbol{x},\boldsymbol{z}))$ for $\boldsymbol{x},\boldsymbol{z} \in X, a,b \in \mathbb{R}^+$ are also kernels on X. This suggests a variable-length representation in which the individuals encode expressions that evaluate to kernel functions.

Given these different search spaces, it is not surprising that the aspects of all major branches of evolutionary computation have been used in evolutionary kernel learning: genetic algorithms (Fröhlich et al. 2004), genetic programming (Howley and Madden 2005; Gagné et al. 2006), evolution strategies (Igel 2005; Friedrichs and Igel 2005), and evolutionary programming (Runarsson and Sigurdsson 2004).

In general, kernel methods assume that the kernel (or at least the ▸ Gram matrix in the training process) is ▸ positive semidefinite (psd). Therefore, it is advisable to restrict the search space such that only psd functions evolve. Other ways of dealing with the problem of ensuring positive semidefiniteness are to assign lethal fitness values to individuals not encoding proper kernels or to construct a psd Gram matrix from the matrix $\boldsymbol{M}$ induced by the training data and a non-psd "kernel" function. The latter can be achieved by subtracting the smallest eigenvalue of $\boldsymbol{M}$ from its diagonal entries.

Gaussian Kernels

Gaussian kernel functions are prevalent. Their general form is $k(\boldsymbol{x},\boldsymbol{z}) = \exp\left(-(\boldsymbol{x}-\boldsymbol{z})^{\mathrm{T}} \boldsymbol{A}(\boldsymbol{x}-\boldsymbol{z})\right)$ for $\boldsymbol{x},\boldsymbol{z} \in \mathbb{R}^n$ and symmetric positive definite (pd) matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$. When adapting $\boldsymbol{A}$, the issue of ensuring that the optimization algorithm only generates pd matrices $\boldsymbol{A}$ arises. This can be achieved by an appropriate parametrization of $\boldsymbol{A}$. Often the search is restricted to matrices of the form $\gamma \boldsymbol{I}$, where $\boldsymbol{I}$ is the unit matrix and $\gamma \in \mathbb{R}^+$ is the only adjustable parameter. However, allowing more flexibility has proven to be beneficial in certain applications (e.g., see Chapelle et al. 2002; Friedrichs and Igel 2005; Glasmachers and Igel 2005). It is straightforward to consider diagonal matrices with positive elements to allow for independent scaling factors weighting the input components. However, only by dropping this restriction one can achieve invariance against both rotation and scaling of the input space. A real-valued encoding that maps onto the set of all symmetric pd matrices can be used such that all modifications of the parameters result in feasible kernels, see the articles by Friedrichs and Igel (2005), Glasmachers and Igel (2005), and Suttorp and Igel (2006) for different parametrizations.

Optimizing Additional Hyperparameters

One of the advantages of evolutionary kernel learning is that it can be easily combined with an optimization of additional hyperparameters of the kernel method. The most prominent example is to encode not only the kernel but also the regularization parameter in evolutionary model selection for SVMs.

Application Example

Notable applications of evolutionary kernel learning include the design of classifiers in bioinformatics (Mersch et al. 2007; Pang and Kasabov 2004; Shi et al. 2004). Let us consider the work by Mersch et al. (2007) as an instructive example. Here, the parameters of a sequence kernel are evolved to improve the prediction of gene starts in DNA sequences. The kernel can be viewed as a weighted sum of 64 kernels, each measuring similarity with respect to a particular trinucleotide sequence (codon). The 64 weights $w_1,\ldots,w_{64}$ are optimized together with an addi-

tional global kernel parameter σ and a regularization parameter C for the SVM. Each individual stores $\boldsymbol{x} \in \mathbb{R}^{66}$, where $(w_1, \ldots, w_{64}, \sigma, C)^{\mathrm{T}} = (\exp(x_1), \ldots, \exp(x_{64}), |x_{65}|, |x_{66}|)^{\mathrm{T}}$. An evolution strategy is applied, using additive multivariate Gaussian mutation and weighted global recombination for variation and rank-based selection. The fitness is determined by five fold cross-validation. The evolved kernels lead to higher classification rates, and the adapted weights reveal the importance of particular codons for the task at hand.

Cross-References

▸ Neuroevolution

Recommended Reading

Chapelle O, Vapnik V, Bousquet O, Mukherjee S (2002) Choosing multiple parameters for support vector machines. Mach Learn 46(1):131–159

Eads DR, Hill D, Davis S, Perkins SJ, Ma J, Porter RB et al (2002) Genetic algorithms and support vector machines for time series classification. In: Bosacchi B, Fogel DB, Bezdek JC (eds) Applications and science of neural networks, fuzzy systems, and evolutionary computation V. Proceedings of the SPIE, vol 4787. SPIE–The International Society for Optical Engineering, Bellington, pp 74–85

Friedrichs F, Igel C (2005) Evolutionary tuning of multiple SVM parameters. Neurocomputing 64(C):107–117

Fröhlich H, Chapelle O, Schölkopf B (2004) Feature selection for support vector machines using genetic algorithms. Int J Artif Intell Tools 13(4):791–800

Gagné C, Schoenauer M, Sebag M, Tomassini M (2006) Genetic programming for kernel-based learning with co-evolving subsets selection. In: Runarsson TP, Beyer H-G, Burke E, Merelo-Guervós JJ, Whitley LD, Yao X (eds) Parallel problem solving from nature (PPSN IX). LNCS, vol 4193. Springer, Berlin, pp 1008–1017

Glasmachers T, Igel C (2005) Gradient-based adaptation of general Gaussian kernels. Neural Comput 17(10):2099–2105

Howley T, Madden M (2005) The genetic kernel support vector machine: description and evaluation. Artif Intell Rev 24(3):379–395

Igel C (2005) Multi-objective model selection for support vector machines. In: Coello Coello CA, Zitzler E, Hernandez Aguirre A (eds) Proceedings of the third international conference on evolutionary multi-criterion optimization (EMO 2005). LNCS, vol 3410. Springer, Berlin, pp 534–546

Igel C (2013) A note on generalization loss when evolving adaptive pattern recognition systems. IEEE Transact Evol Comput 17(3):345–352

Jong K, Marchiori E, van der Vaart A (2004) Analysis of proteomic pattern data for cancer detection. In: Raidl GR, Cagnoni S, Branke J, Corne DW, Drechsler R, Jin Y et al (eds) Applications of evolutionary computing. LNCS, vol 3005. Springer, Berlin, pp 41–51

Mersch B, Glasmachers T, Meinicke P, Igel C (2007) Evolutionary optimization of sequence kernels for detection of bacterial gene starts. Int J Neural Syst 17(5):369–381

Miller MT, Jerebko AK, Malley JD, Summers RM (2003) Feature selection for computer-aided polyp detection using genetic algorithms. In: Clough AV, Amini AA (eds) Medical imaging 2003: physiology and function: methods, systems, and applications. Proceedings of the SPIE, Santa Clara, vol 5031, pp 102–110

Pang S, Kasabov N (2004) Inductive vs. transductive inference, global vs. local models: SVM, TSVM, and SVMT for gene expression classification problems. In: International joint conference on neural networks (IJCNN 2004), vol 2. IEEE Press, Washington, DC, pp 1197–1202

Runarsson TP, Sigurdsson S (2004) Asynchronous parallel evolutionary model selection for support vector machines. Neural Inf Process – Lett Rev 3(3):59–68

Shi SYM, Suganthan PN, Deb K (2004) Multiclass protein fold recognition using multi-objective evolutionary algorithms. In: IEEE symposium on computational intelligence in bioinformatics and computational biology. IEEE Press, Washington, DC, pp 61–66

Suttorp T, Igel C (2006) Multi-objective optimization of support vector machines. In: Jin Y (ed) Multi-objective machine learning. Studies in computational intelligence, vol 16. Springer, Berlin, pp 199–220

Evolutionary Robotics

Phil Husbands
Department of Informatics, Centre for Computational Neuroscience and Robotics, University of Sussex, Brighton, UK

Abstract

Evolutionary robotics uses evolutionary search methods to fully or partially design robotic systems, including their control

systems and sometimes their morphologies and sensor/actuator properties. Such methods are used in a range of ways from the fine-tuning or optimization of established designs to the creation of completely novel designs. There are many applications of evolutionary robotics from wheeled to legged to swimming to flying robots. A particularly active area is the use of evolutionary robotics to synthesize embodied models of complete agent behaviors in order to help explore and generate hypotheses in neurobiology and cognitive science.

Synonyms

Embodied evolutionary learning; Evolution of agent behaviors; Evolution of robot control

Definition

Evolutionary robotics involves the use of ▸ evolutionary computing techniques to automatically develop some or all of the following properties of a robot: the control system, the body morphology, and the sensor and motor properties and layout. Populations of artificial genomes (usually lists of characters and numbers) encode properties of autonomous mobile robots required to carry out a particular task or to exhibit some set of behaviors. The genomes are mutated and interbred creating new generations of robots according to a Darwinian scheme in which the fittest individuals are most likely to produce offspring. Fitness is measured in terms of how good a robot's behavior is according to some evaluation criteria; this is usually automatically measured but may, in the manner of eighteenth-century pig breeders, be based on the experimenters' judgment.

Motivation and Background

Turing's (1950) paper, *Computing Machinery and Intelligence*, is widely regarded as one of the seminal works in artificial intelligence. It is best known for what came to be called the Turing test – a proposal for deciding whether or not a machine is intelligent. However, tucked away toward the end of Turing's wide-ranging discussion of issues arising from the test is a far more interesting proposal. He suggests that worthwhile intelligent machines should be adaptive and should learn and develop but concedes that designing, building, and programming such machines by hand is probably completely infeasible. He goes on to sketch an alternative way of creating machines based on an artificial analog of biological evolution. Each machine would have hereditary material encoding its structure, mutated copies of which would form offspring machines. A selection mechanism would be used to favor better adapted machines – in this case, those that learned to behave most intelligently. Turing proposed that the selection mechanism should largely consist of the experimenter's judgment.

It was not until more than 40 years after their publication that Turing's long forgotten suggestions became reality. Building on the development of principled evolutionary search algorithm by, among others, Holland (1975), researchers at CNR, Rome, Case Western University, the University of Sussex, EPFL, and elsewhere independently demonstrated methodologies and practical techniques to evolve, rather than design, the control systems for primitive autonomous intelligent machines (Beer and Gallagher 1992; Cliff et al. 1993; de Garis 1990; Floreano and Mondada 1994; Husbands and Harvey 1992; Parisi and Nolfi 1993). Thus, the field of *Evolutionary Robotics* was born in the early 1990s. Initial motivations were similar to Turing's: the hand design of intelligent adaptive machines intended for operation in natural environments is extremely difficult, would it be possible to wholly or partly automate the process?

Today, the field of evolutionary robotics has expanded in scope to take in a wide range of applications, including promising new work on autonomous flying machines (Floreano et al. 2008; Vargas et al. 2014; Shim and Husbands 2007), as well as research aimed at exploring specific sci-

entific issues – for instance, principles from neuroscience or questions in cognitive science (Harvey et al. 2005; Philippides et al. 2005; Floreano et al. 2008; Husbands et al. 2014). Such work is able to exploit the fact that evolutionary robotics operates with fewer assumptions about neural architectures and behavior-generating mechanisms than other methods; this means that whole general classes of designs and processes can be explored.

Structure of the Learning System

The key elements of the evolutionary robotics approach are the following:

- An artificial genetic encoding specifying the robot control systems, body plan, sensor properties, etc., along with a mapping to the target system
- A method for measuring the fitness of the robot behaviors generated from these genotypes
- A way of applying selection and a set of "genetic" operators to produce the next generation from the current

The structure of the overall evolutionary process is captured in Fig. 1. The general scheme is like that of any application of an evolutionary search algorithm. However, many details of specific parts of the process, particularly the evaluation step, are peculiar to evolutionary robotics.

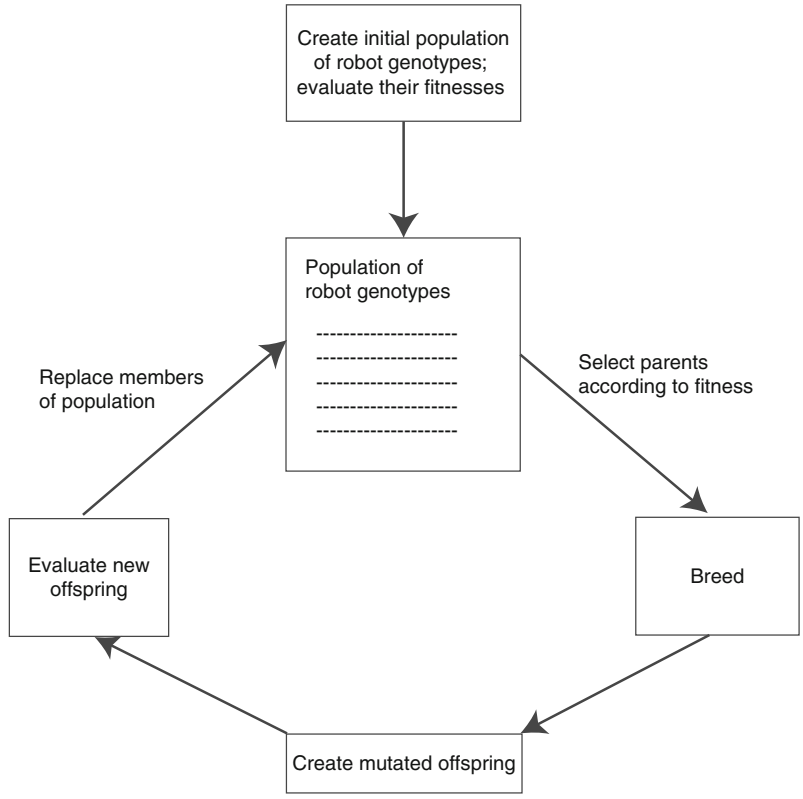
The more general parts of the evolutionary process (selection, breeding, genetic operators such as mutation and crossover, replacement, and population structure) are also found in most other applications of evolutionary computing, and, just as in those other applications, there are many well-documented ways of implementing each (De Jong 2006; Eiben and Smith 2003). Hence, this section focuses on genetic encoding and evaluation as a route to more evolutionary robotics-specific issues. For a much fuller treatment of the subject, see Vargas et al. (2014), Doncieux et al. (2011), Floreano et al. (2008), and Nolfi and Floreano (2000).

Genetic Encoding

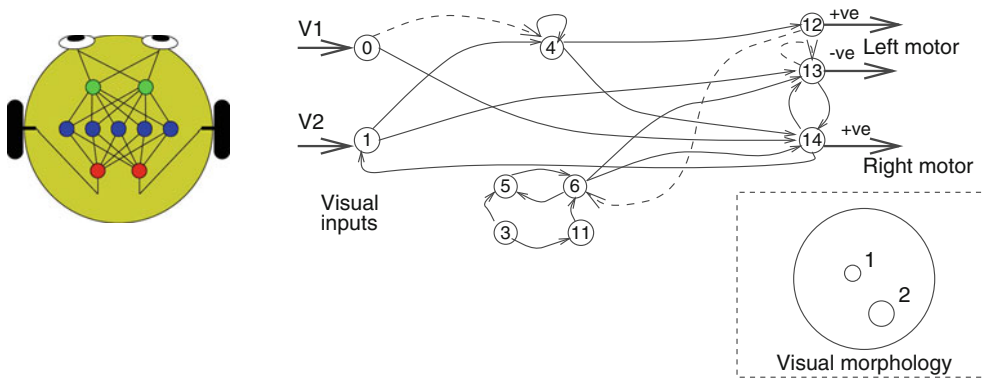
While, as already mentioned, many aspects of the robot design can potentially be under genetic control, *at least* the control system always is. By far the most popular form of controller is some sort of neural network. These range from straightforward feedforward networks of simple elements (Floreano and Mondada 1994) to relatively complex, dynamic, and plastic recurrent networks (Beer and Gallagher 1992; Floreano and Urzelai 2000; Philippides et al. 2005), as illustrated in Fig. 2. In the simplest case, a fixed architecture network is used to control a robot whose sensors feed into the network which in turn feeds out to the robot motors. In this scenario, the parameters of the network (connection weights and relevant properties of the units such as thresholds or biases) are coded as a fixed length string of numerical values.

A more complex case, which has been explored since the very early days of evolutionary robotics (Cliff et al. 1993), involves the evolution of the network architecture as well as the properties of the connections and units. Typically, the size of the network (number of units and connections) and its architecture (wiring diagram) are unconstrained and free to evolve. This involves more complex encodings which can grow and shrink, as units and connections are added or lost, while allowing a coherent decoding of connections between units. These range from relatively simple strings employing blocks of symbols that encode a unit's properties and connections relative to other units (Cliff et al.) to more indirect schemes that make use of developmental, growth processes in some geometric or topological space (Philippides et al. 2005; Stanley et al. 2009) or employ genetic programming-like tree representations in which whole subbranches can be added, deleted, or swapped over (Gruau 1995).

The most general case involves the encoding of control network and body and sensor properties. Various kinds of developmental schemes have been used to encode the construction of body morphologies from basic building blocks, both in simulation and in the real world. The position and properties of sensors can also be



Evolutionary Robotics, Fig. 1 General scheme employed in evolutionary robotics



Evolutionary Robotics, Fig. 2 Evolved neurocontrollers. On the *left* a simple fixed architecture feedforward network is illustrated. The connection weights, and sometimes the neuron properties, are put under evolutionary control. On the *right* a more complex architecture is illustrated. In this case, the whole architecture, including the number of neurons and connections, is under evolutionary control, along with connection and neuron properties and the morphology of a visual sensor that feeds into the network

put under evolutionary control. Sometimes one complex encoding scheme is used for all aspects of the robot under evolutionary control, and sometimes the different aspects are put on separate genotypes.

Fitness Evaluation

The fitness of members of the population is measured, via an evaluation mechanism, in terms of the robot behaviors produced by the control system or control system plus robot morphol-

ogy that it encodes. Fitness evaluation, therefore, consists of translating the genome in question into a robot instantiation and then measuring the aspects of the resulting behavior. In the earliest work aimed at using evolutionary techniques to develop neurocontrollers for particular physical robots, members of a population were downloaded in turn onto the robot and their behavior was monitored and measured either automatically by clever experimental setups (Floreano and Mondada 1994; Harvey et al. 1994) or manually by an observer (Gruau and Quatramaran 1997). The machinery of the evolutionary search algorithm was managed on a host computer, while the fitness evaluations were undertaken on the target robot.

One drawback of evaluating fitness on the robot is that this cannot be done any quicker than in real time, making the whole evolutionary process rather slow. However, in the early work in the field, this approach was taken because it was felt that it was unlikely that simulations could be made accurate enough to allow proper transfer of evolved behavior onto the real robot. However, a careful study of accurate physics-based simulations of a Khepera robot, with various degrees of noise added, proved this assumption false (Jakobi et al. 1995). This led to the development of Jakobi's minimal simulation methodology (Jakobi 1998a), whereby computationally very efficient simulations are built by modeling only those aspects of the robot–environment interaction deemed important to the desired behavior and masking everything else with carefully structured noise (so that evolution could not come to rely on any of those features). These ultrafast, ultralean simulations have successfully been used with many different forms of robot and sensing, with very accurate transfer of behavior from simulation to reality. An alternative approach uses plastic controllers that further adapt through self-organization to help smooth out the differences between an inaccurate simulation and the real world (Urzelai and Floreano 2001). Instead of evolving connection weights, in this approach "learning rules" for adapting connection strengths are evolved – this results in controllers that continually adapt to changes in their environment. For details of further approaches, see Floreano et al. (2008). Much evolutionary robotics work now makes use of simulations; without them it would be impossible to do the most ambitious work on the concurrent evolution of controllers and body morphology (Lipson and Pollack 2000) (to be briefly described later). However, although simulation packages and techniques have developed rapidly in the past few years, there will still inevitably be discrepancies between simulation and reality, and the lessons and insights of the work outlined above should not be forgotten.

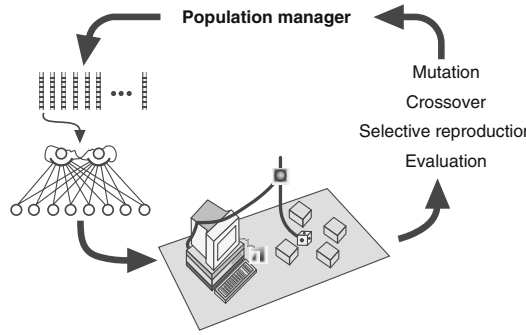
An interesting distinction can be made between implicit and explicit fitness functions in evolutionary robotics (Nolfi and Floreano 2000). In this context, an explicit fitness function rewards specific behavioral elements – such as traveling in a straight line – and hence shapes the overall behavior from a set of specific behavioral primitives. Implicit fitness functions operate at a more indirect, abstract level – fitness points are given for completing some task but they are not tied to specific behavioral elements. Implicit fitness functions might involve components such as maintaining energy levels or covering as much ground as possible, components that can be achieved in many different ways. In practice, it is quite possible to define a fitness function that has both explicit and implicit elements. Often fitness entails multiple and potentially conflicting elements, so methods from multi-objective optimisation have been introduced by some researchers, which can also encourage diversity in robot behavior (Mouret and Doncieux 2012).

Advantages

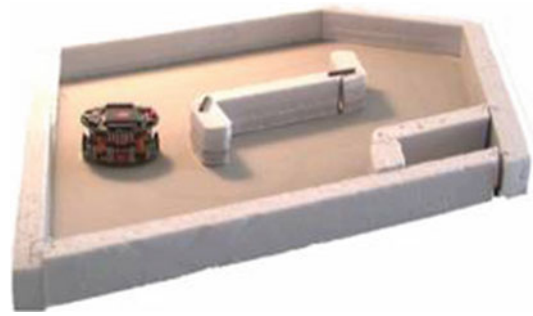
Potential advantages of this methodology include:

- The ability to explore potentially unconstrained designs that have large numbers of free variables. A *class* of robot systems (to be searched) is defined rather than specific, fully defined robot designs. This means fewer assumptions and constraints are necessary in specifying a viable solution.

- The ability to use the methodology to fine-tune the parameters of an already successful design.
- The ability, through the careful design of fitness criteria and selection techniques, to take into account multiple, and potentially conflicting, design criteria and constraints (e.g., efficiency, cost, weight, power consumption, etc.).
- The possibility of developing highly unconventional and minimal designs.
- The ability to explicitly take into account robustness and reliability as major driving force behind the fitness measure, factors that are particularly important for certain applications.



Evolutionary Robotics, Fig. 3 Setup for early EPFL evolutionary robotics experiments with the Khepera robot (see text for details). Used with permission



Evolutionary Robotics, Fig. 4 The simple environment used for evolving obstacle avoidance behaviors with a Khepera robot. Used with permission

Applications

For a detailed survey of applications of evolutionary robotics, see Floreano et al. (2008) and Vargas et al. (2014); this section gives a brief overview of some areas covered by the methodology to give a better idea of the techniques involved and to indicate the scope of the field.

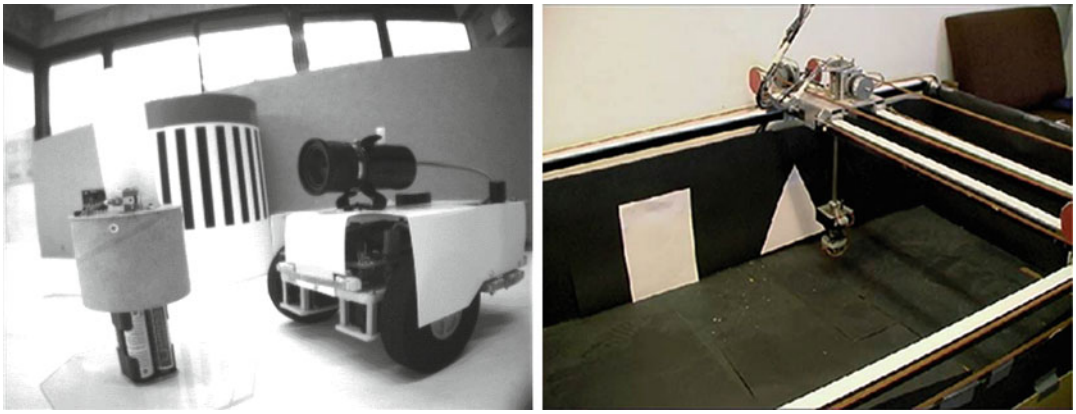
Prominent early centers for research in this area were EPFL and Sussex University, both of which are still very active in the field. Much of the early EPFL work used the miniature Khepera robot (Mondada et al. 1993), which became a popular tool in many areas of robotics research. In its simplest form, it is a two-wheeled cylindrical robot with a ring of IR sensors around its body. The first successful evolutionary robotics experiments at EPFL employed the setup illustrated in Figs. 3 and 4. A population of bit strings encoded the connection weights and node thresholds for a simple fixed architecture feedforward neural network. Each member of the population was decoded into a particular instantiation of a neural network controller which was then downloaded onto the robot (Floreano and Mondada 1994). This controlled the robot for a fixed period of time as it moved around the environment shown in Fig. 4.

The following simple fitness function was used to evolve obstacle avoidance behaviors:

$$F = V + (1 - \sqrt{DV}) + (1 - I)$$

where V is the average rotation speed of opposing wheels, DV is the difference between signed speed values of opposing wheels, and I is the activation value of the IR sensor with the highest input (readings are high if an obstacle is close to a sensor). Maximizing this function ensures high speed, a tendency to move in straight lines, and avoidance of walls and obstacles in the environment. After about 36 h of real-world evolution using this setup, controllers were evolved that successfully generated efficient motion around the course, avoiding collisions with the walls.

At the same time as this work was going on at EPFL, a series of pioneering experiments on evolving visually guided behaviors were being performed at Sussex University (Cliff et al. 1993; Harvey et al. 1994) in which discrete-



Evolutionary Robotics, Fig. 5 An early version of the Sussex gantry robot (*right*) was a "hardware simulation" of a robot such as that shown on the *left*. It allowed real-world evolution of visually guided behaviors in an easily controllable experimental setup (see text for further details)

time dynamical recurrent neural networks and visual sampling morphologies were concurrently evolved to allow a gantry robot (as well as other more standard mobile robots) to perform various visually guided tasks. An early instantiation of the Sussex gantry robot is shown in Fig. 5.

A CCD camera points down toward a mirror angled at 45°. The mirror can rotate around an axis perpendicular to the camera's image plane. The camera is suspended from the gantry allowing motion in the X, Y, and Z dimensions. This effectively provides an equivalent to a wheeled robot with a forward facing camera when only the X and Y dimensions of translation are used (see Fig. 5).

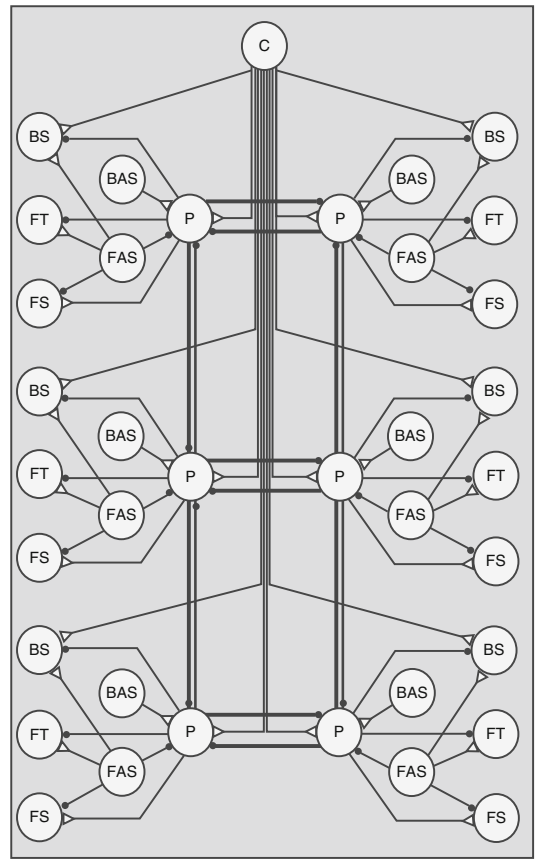
The apparatus was initially used in a manner similar to the real-world EPFL evolutionary robots setup illustrated in Fig. 3. A population of strings encoding robot controllers and visual sensing morphologies are stored on a computer to be downloaded one at a time onto the robot. The exact position and orientation of the camera head can be accurately tracked and used in the fitness evaluations. A number of visually guided navigation behaviors were successfully achieved, including navigating around obstacles and discriminating between different objects. In the experiment illustrated in Fig. 5, starting from a random position and orientation, the robot has to move to the triangle rather than the rectangle. This has to be achieved irrespective of the relative positions of the shapes and under very noisy lighting conditions. The architecture and all parameters of recurrent neural network controllers were evolved in conjunction with visual sampling morphologies – only genetically specified patches from the camera image were used (by being fed to input neurons according to a genetic specification), the rest of the image is thrown away. This resulted in extremely minimal systems only using two or three pixels of visual information yet still able to very robustly perform the task under highly variable lighting conditions. Behaviors were evolved in an incremental way, with more complex capabilities being evolved from populations of robots that were successful at some simpler task (for details see Harvey et al. 1994 and Harvey et al. 1997). The highly minimal yet very robust systems developed highlighted the potential for evolutionary robotics techniques in areas such as space exploration where there is a great pressure to minimize resources while maintaining reliability (Hobbs et al. 1996).

Since this early work, many different behaviors have been successfully evolved on a wide range of robots (Floreano et al. 2008; Nolfi and Floreano 2000; Vargas et al. 2014; Doncieux et al. 2011). There is not enough room to give an adequate summary of the whole field, so a few interesting subareas are highlighted below.

Over the past 15 years or so, there has been a growing body of work on evolving controllers

for various kinds of walking robots – a nontrivial sensorimotor coordination task. Early work in this area concentrated on evolving dynamical network controllers for simple simulated insects (often inspired by cockroach studies), which were required to walk in uncomplicated environments (e.g., de Garis 1990; Beer and Gallagher 1992). The promise of this work soon led to versions of this methodology being used on real robots. Probably, the first success in this direction was by Lewis et al. (1992) who evolved a neural controller for a simple hexapod robot, using coupled oscillators built from continuous-time, leaky-integrator, artificial neurons. The robot was able to execute an efficient tripod gait on flat surfaces. All evaluations were done on the actual robot with each leg connected to its own pair of coupled neurons, leg swing being driven by one neuron, and leg elevation by the other. These pairs of neurons were cross-connected, in a manner similar to that used in the neural architecture shown in Fig. 6, to allow coordination between the legs. This architecture for locomotion, introduced by Beer et al. (1989), was based on the studies of cockroaches and has been much used ever since. Gallagher et al. (1996) used a generalization of it to evolve controllers for generating locomotion in a hexapod robot. This machine was more complex than Lewis et al.'s, with a greater number of degrees of freedom per leg. In this work, each leg was controlled by a fully connected network of five continuous-time, leaky-integrator neurons, each receiving a weighted sensory input from that leg's angle sensor. The connection weights and neuron time constants and biases were under genetic control. This produced efficient tripod gaits for walking on flat surfaces. In order to produce a wider range of gaits operating at a number of speeds such that rougher terrain could be successfully negotiated, a slightly different distributed architecture, more inspired by stick insect studies, was found to be more effective (Beer et al. 1997).

Jakobi (1998b) successfully used his minimal simulation techniques to evolve controllers for an eight-legged robot. Evolution in simulation took less than 2 h on what would today be regarded as a very slow computer and then transferred successfully to the real robot. Jakobi evolved modular controllers based on Beer's continuous recurrent network architecture to control the robot as it engaged in walking about its environment, avoiding obstacles and seeking out goals. The robot could smoothly change gait, move backward and forward, and even turn on the spot. More recently, related approaches have been successfully used to evolve controllers for more mechanically sophisticated robots such as the Sony Aibo (Tllez et al. 2006). In the last few years, there has also been successful work on evolving coupled oscillator style neural



Evolutionary Robotics, Fig. 6 Schematic diagram of a distributed neural network for the control of locomotion as used by Beer et al. Excitatory connections are denoted by *open triangles*, and inhibitory connections are denoted by *filled circles*. *C*, command neuron; *P*, pacemaker neuron; *FT*, foot motor neuron; *FS* and *BS*, forward swing and backward swing motor neurons; *FAS* and *BAS*, forward and backward angle sensors. Reproduced with permission

controllers for the highly unstable dynamic problem of biped walking. Reil and Husbands (2002) showed that accurate physics-based simulations using physics engine software could be used to develop controllers able to generate successful bipedal gaits. Reil and colleagues have now significantly developed this technology to exploit its commercial possibilities in the animation and games industries (see www.naturalmotion.com for further details). Vaughan has taken related work in another direction. He has successfully applied evolutionary robotics techniques to evolve a simulation of a 3D ten degree of freedom bipedal robot. This machine demonstrates many of the properties of human locomotion. By using passive dynamics and compliant tendons, it conserves energy while walking on a flat surface. Its speed and gait can be dynamically adjusted and it is capable of adapting to discrepancies in both its environment and its body's construction (Vaughan et al. 2004). In general, the evolutionary development of neural network walking controllers, with their intricate dynamics, produces a wider range of gaits and generates smoother, more adaptive locomotion than the more standard use of finite state machine-based systems employing parameterized rules governing the timing and coordination of individual leg movements.

Early single robot research was soon expanded to handle interactions between multiple robots. Floreano and Nolfi did pioneering work on the coevolution of predator–prey behaviors in physical robots (Floreano et al. 2007). The fitness of the prey robot was measured by how quickly it could catch the prey; the fitness of the prey was determined by how long it could escape the predator. Two Khepera robots were used in this experiment, each had the standard set of proximity sensors but the predator also has a vision system, and the prey was able to move twice as fast as the predator. A series of interesting chasing and evasion strategies emerged. Later Quinn et al. (2003) demonstrated the evolution of coordinated cooperative behavior in a group of robots. A group of robots equipped only with IR proximity sensors were required to move as far as possible as a coordinated group starting from a random configuration. The task was solved by the robots adopting and then maintaining a specific formation. Analysis of the best evolved solution showed that it involved the robots adopting different roles, with the identical robots collectively "deciding" which robot would perform each role. Given the minimal sensing constraints, the evolved system would have proved extremely difficult to have designed by hand. For discussion of other multiple robot behaviors, see Floreano et al. (2008) and Vargas et al. (2014).

In the work described so far, control systems have been evolved for preexisting robots: the brain is constrained to fit a particular body and set of sensors. Of course in nature, the nervous system evolved simultaneously with the rest of the organism. As a result, the nervous system is highly integrated with the sensory apparatus and the rest of the body: the whole operates in a harmonious and balanced way – there are no distinct boundaries between the control system, the sensors, and the body.

Karl Sims started to explore the concurrent evolution of the brain and the body in his highly imaginative work involving simulated 3D "creatures" (Sims 1994). In this work, the creatures coevolved under a competitive scenario in which they were required to try and gain control of a resource (a cube) placed in the center of an arena. Both the morphology of the creatures and the neural system controlling their actuators were under evolutionary control.

Lipson and Pollack (2000), working at Brandeis University, pushed the idea of fully evolvable robot hardware about as far as was reasonably technologically feasible at the time. In an important piece of research, directly inspired by Sims' earlier simulation work, autonomous "creatures" were evolved in simulation out of basic building blocks (neurons, plastic bars, and actuators). The bars could connect together to form arbitrary truss structures with the possibility of both rigid and articulated substructures. Neurons could be connected to each other and to the bars whose length they would then control via a linear actuator. Machines defined in this way were required to move as far as possible in a limited time. The fittest individuals were then fabricated robotically

Evolutionary Robotics, Fig. 7 A fully automatically evolved robot developed on the Golem project (see text for details). Used with permission

using rapid manufacturing technology (plastic extrusion 3D printing) to produce results such as that shown in Fig. 7. They thus achieved autonomy of design and construction using evolution in a "limited universe" physical simulation coupled to automatic fabrication. The highly unconventional designs thus realized performed as well in reality as in simulation. The success of this work points the way to new possibilities in developing energy-efficient fault-tolerant machines.

Pfeifer and colleagues at Zurich University have explored issues central to the key motivation for fully evolvable robot hardware: the balanced interplay between body morphology, neural processing, and environment in the generation of adaptive behavior, and have developed a set of design principles for intelligent systems in which these issues take center stage (Pfeifer and Bongard 2007). Examples of interesting current work in this direction includes (Bongard 2011; Johnson et al. 2014).

Future Directions

Major ongoing challenges – methodological, theoretical, and technological – include finding the best way to incorporate development and lifetime plasticity within the evolutionary framework (this involves trends coming from the emerging field of epigenetic robotics), understanding better what the most useful building blocks are for evolved neurocontrollers, and finding efficient ways to scale work on concurrently evolving bodies and brains, especially in an open-ended way in the real world. For some grand challenges in the field, see (Eiben 2014).

There are very interesting developments in the evolution of group behaviors and the emergence of communication (Di Paolo 1998; Floreano et al. 2007; Quinn 2001; Vargas et al. 2014), the use of evolutionary robotics as a tool to illuminate problems in cognitive science (Beer 2003; Harvey et al. 2005) and neuroscience (Di Paolo 2003; Philippides et al. 2005; Seth 2005; Husbands et al. 2014), in developing flying behaviors (Floreano et al. 2007; Shim and Husbands 2007; Vargas et al. 2014), and in robots that have some form of self-model (Bongard et al. 2006), to name but a few.

Cross-References

▸ Neuroevolution

Recommended Reading

Beer RD (2003) The dynamics of active categorical perception in an evolved model agent (with commentary and response). Adapt Behav 11(4): 209–243

Beer RD, Chiel HJ, Sterling LS (1989) Heterogeneous neural networks for adaptive behavior in dynamic environments. In: Touretzky D (ed) Neural information processing systems, vol 1. Morgan Kauffman, San Francisco, pp 577–585

Beer RD, Gallagher JC (1992) Evolving dynamical neural networks for adaptive behaviour. Adapt Behav 1:94–110

Beer RD, Quinn RD, Chiel HJ, Ritzmann RE (1997) Biologically-inspired approaches to robotics. Commun ACM 40(3):30–38

Bongard J (2011) Morphological change in machines accelerates the evolution of robust behavior. Proc Natl Acad Sci 108(4):1234–1239

Bongard J, Zykov V, Lipson H (2006) Resilient machines through continuous self-modeling. Science 314:1118–1121

Cliff D, Harvey I, Husbands P (1993) Explorations in evolutionary robotics. Adapt Behav 2:73–110

de Garis H (1990) Genetic programming: evolution of time dependent neural network modules which teach a pair of stick legs to walk. In: Proceedings of the 9th European conference on artificial intelligence, Stockholm, pp 204–206

De Jong KA (2006) Evolutionary computation: a unified approach. MIT Press, Cambridge
Di Paolo E (1998) An investigation into the evolution of communication. Adapt Behav 6(2):285–324
Di Paolo EA (2003) Evolving spike-timing dependent plasticity for single-trial learning in robots. Philos Trans R Soc A 361:2299–2319
Doncieux S, Bredeche N, Mouret J-B (eds) (2011) New Horizons in evolutionary robotics: extended contributions from the 2009 EvoDeRob workshop. Studies in computational intelligence, vol 341. Springer, Berlin
Eiben AE (2014) Grand challenges for evolutionary robotics. Front Robot AI 1(4). doi:10.3389/frobt.2014.00004
Eiben AE, Smith JE (2003) Introduction to evolutionary computing. Springer, Berlin
Floreano D, Hauert S, Leven S, Zufferey JC (2007) Evolutionary swarms of flying robots. In: Floreano D (ed) Proceedings of the international symposium on flying insects and robots. EPFL, Monte Verita, pp 35–36
Floreano D, Husbands P, Nolfi S (2008) Evolutionary robotics. In: Siciliano B, Khatib O (eds) Springer handbook of robotics (Chap. 61). Springer, Berlin, pp 1423–1451
Floreano D, Mitri S, Magnenat S, Keller L (2007) Evolutionary conditions for the emergence of communication in robots. Curr Biol 17:514–519
Floreano D, Mondada F (1994) Automatic creation of an autonomous agent: genetic evolution of a neural-network driven robot. In: Cliff D, Husbands P, Meyer J, Wilson SW (eds) From animals to animats III: proceedings of the third international conference on simulation of adaptive behavior. MIT Press-Bradford Books, Cambridge, pp 402–410
Floreano D, Nolfi S (1997) Adaptive behavior in competing co-evolving species. In: Husbands P, Harvey I (eds) Proceedings of the 4th European conference on artificial life. MIT Press, Cambridge, pp 378–387
Floreano D, Urzelai J (2000) Evolutionary robots with on-line self-organization and behavioral fitness. Neural Netw 13(4–5):431–443
Gallagher J, Beer R, Espenschiel M, Quinn R (1996) Application of evolved locomotion controllers to a hexapod robot. Robot Auton Syst 19(1):95–103
Gruau F (1995) Automatic definition of modular neural networks. Adapt Behav 3(2):151–183
Gruau F, Quatramaran K (1997) Cellular encoding for interactive evolutionary robotics. In: Husbands P, Harvey I (eds) Proceedings of the 4th European conference on artificial life. The MIT Press/Bradford Books, Cambridge
Harvey I, Di Paolo E, Wood R, Quinn M, Tuci E (2005) Evolutionary robotics: a new scientific tool for studying cognition. Artif Life 11(1–2):79–98
Harvey I, Husbands P, Cliff DT (1994) Seeing the light: artificial evolution, real vision. In: Cliff DT, Husbands P, Meyer JA, Wilson S (eds) From animals to animats 3: proceedings of the third international conference on simulation of adaptive behaviour, SAB94. MIT Press, Cambridge, pp 392–401
Harvey I, Husbands P, Cliff D, Thompson A, Jakobi N (1997) Evolutionary robotics: the Sussex approach. Robot Auton Syst 20:205–224
Hobbs J, Husbands P, Harvey I (1996) Achieving improved mission robustness. In: 4th European Space Agency workshop on advanced space technologies for robot applications – ASTRA'96, Noordwijk, The Netherlands ESTEC
Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
Husbands P, Harvey I (1992) Evolution versus design: controlling autonomous mobile robots. In: Proceedings of 3rd annual conference on artificial intelligence, simulation and planning in high autonomy systems. Computer Society Press, Los Alimitos, pp 139–146
Husbands P, Moioli R, Shim Y, Philippides A, Vargas P, O'Shea M (2014) Evolutionary robotics and neuroscience. In: Vargas P, Di Paolo E, Harvey I, Husbands P (eds.) The horizons of evolutionary robotics. MIT Press, Cambridge, pp 17–63
Jakobi N (1998a) Evolutionary robotics and the radical envelope of noise hypothesis. Adapt Behav 6:325–368
Jakobi N (1998b) Running across the reality gap: octopod locomotion evolved in a minimal simulation. In: Husbands P, Meyer JA (eds) Evolutionary robotics: first European workshop, EvoRobot98. Springer, Berlin, pp 39–58
Jakobi N, Husbands P, Harvey I (1995) Noise and the reality gap: the use of simulations in evolutionary robotics. In: Moran F et al (eds) Proceedings of 3rd European conference on artificial life. Springer, Berlin, pp 704–720
Johnson C, Philippides A, Husbands P (2014) Active shape discrimination with physical reservoir computers. In: Proceedings of Alife 14. MIT Press, Cambridge, pp 178–185
Lewis MA, Fagg AH, Solidum A (1992) Genetic programming approach to the construction of a neural network for a walking robot. In: Proceedings of IEEE international conference on robotics and automation. IEEE Press, Washington, pp 2618–2623
Lipson H, Pollack J (2000) Automatic design and manufacture of robotic lifeforms. Nature 406:974–978
Mondada F, Franzi E, Ienne P (1993) Mobile robot miniaturization: a tool for investigation in control algorithms. In: Yoshikawa T, Miyazaki F (eds) Proceedings of the third international symposium on experimental robotics. Springer, Berlin, pp 501–513
Mouret J-B, Doncieux S (2012) Encouraging behavioral diversity in evolutionary robotics: an empirical study. Evol Comput 20(1):91–133
Nolfi S, Floreano D (2000) Evolutionary robotics: the biology. In: Intelligence, and technology of self-organizing machines. MIT Press/Bradford Books, Cambridge

Parisi D, Nolfi S (1993) Neural network learning in an ecological and evolutionary context. In: Roberto V (ed) Intelligent perceptual systems. Springer, Berlin, pp 20–40
Pfeifer R, Bongard J (2007) How the body shapes the way we think: a new view of intelligence. MIT Press, Cambridge
Philippides A, Husbands P, Smith T, O'Shea M (2005) Flexible couplings: diffusing neuromodulators and adaptive robotics. Artif Life 11(1&2):139–160
Quinn M (2001) Evolving communication without dedicated communication channels. In: Kelemen J, Sosik P (eds) Proceedings of the 6th European conference on artificial life, ECAL'01. Springer, Berlin, pp 357–366
Quinn M, Smith L, Mayley G, Husbands P (2003) Evolving controllers for a homogeneous system of physical robots: structured cooperation with minimal sensors. Philos Trans R Soc Lond Ser A: Math Phys Eng Sci 361:2321–2344
Reil T, Husbands P (2002) Evolution of central pattern generators for bipedal walking in real-time physics environments. IEEE Trans Evol Comput 6(2): 10–21
Seth AK (2005) Causal connectivity analysis of evolved neural networks during behavior. Netw Comput Neural Syst 16(1):35–54
Shim YS, Husbands P (2007) Feathered flyer: integrating morphological computation and sensory reflexes into a physically simulated flapping-wing robot for robust flight Manoeuvre. In: Proceedings of ECAL. LNCS, vol 4648. Springer, Berlin, pp 756–765
Sims K (1994) Evolving 3D morphology and behavior by competition. In: Brooks R, Maes P (eds) Proceedings of artificial life IV. MIT Press, Cambridge, pp 28–39
Stanley K, D'Ambrosio D, Gauci J (2009) A hypercube-based encoding for evolving large-scale neural networks. Artif Life 15(2):185–212
Tllez R, Angulo C, Pardo D (2006) Evolving the walking behaviour of a 12 DOF quadruped using a distributed neural architecture. In: 2nd international workshop on biologically inspired approaches to advanced information technology (Bio-ADIT'2006). LNCS, vol 385. Springer, Berlin, pp 5–19
Turing AM (1950) Computing machinery and intelligence. Mind 59:433–460
Urzelai J, Floreano D (2001) Evolution of adaptive synapses: robots with fast adaptive behavior in new environments. Evol Comput 9:495–524
Vargas P, Di Paolo E, Harvey I, Husbands P (2014) The horizons of evolutionary robotics. MIT Press, Cambridge
Vaughan E, Di Paolo EA, Harvey I (2004) The evolution of control and adaptation in a 3D powered passive dynamic walker. In: Pollack J, Bedau M, Husbands P, Ikegami T, Watson R (eds) Proceedings of the ninth international conference on the simulation and synthesis of living systems artificial life IX. MIT Press, Cambridge, pp 139–145

Evolving Neural Networks

▸ Neuroevolution

Example

▸ Instance

Example Space

▸ Instance Space

Example-Based Programming

▸ Inductive Programming

Expectation Maximization Clustering

Xin Jin[1] and Jiawei Han[2]
[1]PayPal Inc., San Jose, CA, USA
[2]University of Illinios at Urbana-Champaign, Urbana, IL, USA

Abstract

The expectation maximization (EM) based clustering is a probabilistic method to partition data into clusters represented by model parameters. Extensions to the basic EM algorithm include but not limited to the stochastic EM algorithm (SEM), the simulated annealing EM algorithm (SAEM), and the Monte Carlo EM algorithm (MCEM).

Synonyms

Mixture model

Definition

The expectation maximization (EM) algorithm (Dempster et al. 1977; Fraley and Raftery 1998) finds maximum likelihood estimates of parameters in probabilistic models. EM is an iterative method which alternates between two steps, expectation (E) and maximization (M). For clustering, EM makes use of the finite Gaussian mixtures model and estimates a set of parameters iteratively until a desired convergence value is achieved. The mixture is defined as a set of K probability distributions and each distribution corresponds to one cluster. An instance is assigned with a membership probability for each cluster.

The EM algorithm for partitional clustering works as follows:

1. Guess initial parameters of the models: mean and standard deviation (if using normal distribution model).
2. Iteratively refine the parameters with E and M steps. In the E step: compute the membership possibility for each instance based on the initial parameter values. In the M step: recompute the parameters based on the new membership possibilities.
3. Assign each instance to the cluster with which it has highest membership possibility.

Refer to Celeux and Govaert (1995) for details about the E and M steps for multivariate normal mixture models parameterized via the eigenvalue decomposition.

The EM algorithm for clustering becomes time consuming to compute for models with very large numbers of components, because the number of conditional probabilities associated with each instance is the same as the number of components in the mixture.

Extensions

There are many extensions to the EM-based clustering algorithm. Celeux et al. (1996) compared three different stochastic versions of the EM algorithm: the stochastic EM algorithm (SEM), the simulated annealing EM algorithm (SAEM), and the Monte Carlo EM algorithm (MCEM). SEM was shown to be efficient for locating significant maxima of the likelihood function. The classification EM (CEM) algorithm (Celeux and Govaert 1992) incorporates a classification step between the E-step and the M-step using a maximum a posteriori (MAP) principle. The K-means algorithm becomes a particular version of the CEM algorithm corresponding to the uniform spherical Gaussian model. Yang et al. (2012) proposed an EM clustering algorithm for Gaussian mixture models, which is robust to initialization and different cluster sizes with a schema to automatically obtain an optimal number of clusters.

Softwares

The following softwares have implementations of the EM clustering algorithm:

- Weka. Open Source Data Mining Software in Java (Hall et al. 2009), from Machine Learning Group at the University of Waikato: http://www.cs.waikato.ac.nz/ml/weka/index.html
- LNKnet Software. Written in C. A public domain software from MIT Lincoln Laboratory: http://www.ll.mit.edu/mission/communications/cyber/softwaretools/lnknet/lnknet.html
- EMCluster (Chen et al. 2012). R package. It provides EM algorithms and several efficient initialization methods for clustering of finite mixture Gaussian distribution with unstructured dispersion in both unsupervised and semi-supervised learning. http://cran.r-project.org/web/packages/EMCluster/

Recommended Reading

Celeux G, Govaert G (1992) A classification em algorithm for clustering and two stochastic versions. Comput Stat Data Anal 14(3):315–332

Celeux G, Govaert G (1995) Gaussian parsimonious clustering models. Pattern Recognit 28(5):781–793
Celeux G, Chauveau D, Diebolt J (1996) Stochastic versions of the em algorithm: an experimental study in the mixture case. J Stat Comput Simul 55(4):287–314
Chen W-C, Maitra R, Melnykov V (2012) EMCluster: EM algorithm for model-based clustering of finite mixture Gaussian distribution. R Package, http://cran.r-project.org/package=EMCluster
Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc Ser B (Methodol) 39(1):1–38
Fraley C, Raftery AE (1998) How many clusters? Which clustering method? Answers via model-based cluster analysis. Comput J 41(8):578–588
Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. ACM SIGKDD Explor Newsl 11(1):10–18
Yang M-S, Lai C-Y, Lin C-Y (2012) A robust em clustering algorithm for Gaussian mixture models. Pattern Recognit 45(11):3950–3961

Expectation Propagation

Tom Heskes
Institute for Computing and Information Sciences, Radboud University Nijmegen, Nijmegen, The Netherlands

Synonyms

EP

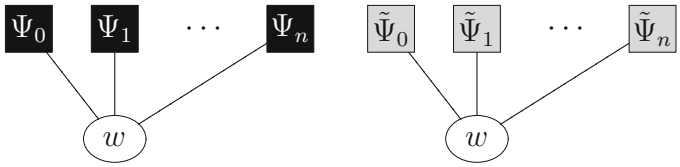
Definition

Expectation propagation is an algorithm for Bayesian machine learning. It tunes the parameters of a simpler approximate distribution (e.g., a Gaussian) to match the exact posterior distribution of the model parameters given the data. Expectation propagation operates by propagating messages, similar to the messages in (loopy) ▸ belief propagation. Whereas messages in belief propagation correspond to exact belief states, messages in expectation propagation correspond to approximations of the belief states in terms of expectations, such as means and variances. It is a deterministic method especially well suited to large databases and dynamic systems, where exact methods for Bayesian inference fail and ▸ Monte Carlo methods are far too slow.

Motivation and Background

One of the main problems for ▸ Bayesian methods is their computational expense: computation of the exact posterior given the observed data typically requires the solution of high-dimensional integrals that have no analytical expressions. Approximation algorithms are needed to approximate this posterior as accurately as possible. These techniques for approximate inference can be subdivided in two categories: deterministic approaches and stochastic sampling (Monte Carlo) methods. Having the important advantage that (under certain conditions) they give exact results in the limit of an infinite number of samples, ▸ Monte Carlo methods are the method of choice in Bayesian statistics. However, in particular, when dealing with large databases, the time needed for stochastic sampling to obtain a reasonably accurate approximation of the exact posterior can be prohibitive. This explains the need for faster, deterministic approaches, such as the Laplace approximation, ▸ variational approximations, and expectation propagation.

Expectation propagation was first described by Thomas Minka in his thesis Minka (2001). It can be viewed as a generalization and reformulation of the earlier ADATAP algorithm of Manfred Opper and Ole Winther (2001). Expectation propagation quickly became one of the most popular deterministic approaches for approximate Bayesian inference. Expectation propagation improves upon assumed density filtering, a classical method from stochastic control, by iteratively refining local approximations instead of computing them just once. Furthermore, it encompasses loopy belief propagation, a popular method for approximate inference in probabilistic graphical models, as a special case. Where loopy belief propagation is restricted to models of discrete variables only, expectation propagation applies to



Expectation Propagation, Fig. 1 (*left-hand side*) A so-called factor graph corresponding to the i.i.d. assumption in Bayesian machine learning. Each box corresponds to a factor or term. A *circle* corresponds to a variable. Factors are connected to the variables that they contain. Ψ_0 corresponds to the prior, and $\Psi_1 \ldots \Psi_n$ are the likelihood terms for the n data points. The *right-hand side* is a factor graph of the approximating distribution. The original terms have been replaced by term approximations

a much wider class of probabilistic ▸ graphical models with discrete and continuous variables and complex interactions between them.

Structure of Learning System

Bayesian Machine Learning

In the Bayesian framework for machine learning, you should enumerate all reasonable models of the data and assign a prior belief $P(w)$ to each of these models w. Then, upon observing the data D, you compute the likelihood $P(D|w)$ to evaluate how probable the data was under each of these models. The product of the prior and the likelihood gives you, up to a normalization constant, the posterior probability $P(w|D)$ over models given the data:

$$P(w|D) = \frac{P(D|w)P(w)}{P(D)} ,$$

where the normalization term $P(D)$ is called the probability of the data or "evidence." This posterior probability incorporates all you have learned from the data D regarding the models w under consideration. As indicated above, exact calculation of this posterior probability is often infeasible, because the normalization term requires the solution of intractable sums or integrals.

In its simplest setting, the data D consists of n observations, $x_1, \ldots, x_n$, which are assumed to be i.i.d. (independent and identically distributed). The posterior probability then factorizes into $n + 1$ terms, one for each observation and one for the prior. With definitions $\Psi_0(w) \equiv P(w)$ and $\Psi_i(w) \equiv P(x_i|w)$, we can rewrite

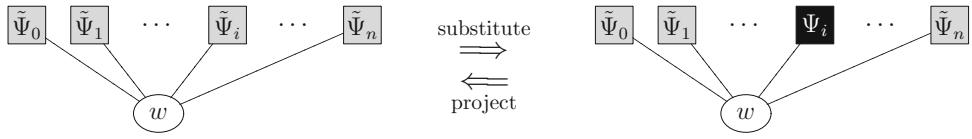
$$P(w|D) = \frac{P(w)\prod_{i=1}^{n} P(x_i|w)}{P(D)} \equiv \frac{\prod_{i=0}^{n} \Psi_i(w)}{P(D)} .$$

This factorization is visualized in the so-called factor graph in Fig. 1. We will use it as a running example in the following.

Assumed Density Filtering

Expectation propagation can be interpreted by an iterative refinement of assumed density filtering. In assumed density filtering, we add terms one by one and project in each step back to the "assumed density." For example, suppose that our prior probability $P(w) = \Psi_0(w)$ is a (known) Gaussian distribution over model parameters w, the terms corresponding to the data points are non-Gaussian, and we aim to find an appropriate Gaussian approximation $Q(w)$ to the exact (non-Gaussian) posterior $P(w|D)$. Our first approximation is the prior itself. Assumed-density filtering now proceeds by adding terms one at a time, where at each step we approximate the resulting distribution as closely as possible by a Gaussian. The pseudo-code is given in Algorithm 1, where $Q_{0:i}(w)$ denotes the approximation obtained after incorporating the prior and the first i observations.

If we use the Kullback-Leibler divergence as the distance measure from the non-Gaussian (but normalized) product of $Q_{0:i-1}(w)$ and $\Psi_i(w)$ and the Gaussian approximation, projection becomes "moment matching": the result of the projection



Expectation Propagation, Fig. 2 Visualization of expectation propagation when recomputing the term approximation for observation i

is the Gaussian that has the same mean and covariance matrix as the non-Gaussian product.

Expectation Propagation

When in assumed density filtering we add the term $\Psi_i(w)$, the Gaussian approximation changes from $Q_{0:i-1}(w)$ to $Q_{0:i}(w)$. We will call the quotient of the two the *term approximation* (here and in the following we ignore normalization constants):

$$\tilde{\Psi}_i(w) = \frac{Q_{0:i}(w)}{Q_{0:i-1}(w)} .$$

In our running example, term approximations are quotients between two different Gaussian densities and therefore have a Gaussian form themselves. Since the prior $\Psi_0(w)$ is a Gaussian density, $\tilde{\Psi}_0(w) = \Psi_0(w)$. The approximation $Q_{0:n}(w)$ is equal to the product of all term approximations and is visualized on the right-hand side of Fig. 1. In assumed density filtering, the resulting approximation depends on the ordering in which the terms have been added. For example, if the terms had been added in reverse order, their term approximations might have been (slightly) different.

Expectation propagation now generalizes assumed density filtering by iteratively refining these term approximations. When successful, the final approximation will be independent of the ordering. Pseudo-code of expectation propagation is given in Algorithm 2. In step 1 through 5, the term approximations are initialized; in step 6 through 12, these term approximations are iteratively refined until they no longer change. In step 8, we take out the previous term approximation from the current approximation. In step 9, we put back in the exact term and project back to a Gaussian, like we did in assumed density filtering. It is easy to check that the approximation $Q(w)$ after the first loop equals the approximation $Q_{0:n}(w)$ obtained with assumed density filtering. The recalculation of the term approximation corresponding to observation i is visualized in Fig. 2.

Computational Aspects

With expectation propagation, we have to do a little more bookkeeping than with assumed density filtering: we have to keep track of the term approximations. One loop of expectation propagation is about as expensive as running assumed density filtering. Typically, about five iterations are sufficient for convergence.

The crucial operation is in step 3 of Algorithm 1 and step 9 of Algorithm 2. Here we have to compute the moments of the (non-Gaussian) probability distribution on the right-hand side. In most cases, we do not have analytical expressions for these moments and have to compute them numerically, e.g., using Gaussian quadrature. We then obtain the moments (mean and covariance matrix) of the new approximation $Q(w)$. Divisions and multiplications correspond to a simple subtraction and addition of so-called canonical parameters. For the Gaussian these canonical parameters are the inverse of the covariance matrix (precision matrix) and the product of the precision matrix and the mean. The bottom line is that we go back and forth between distributions in terms of moments and in terms of canonical parameters. For a Gaussian, this requires computing the inverse of the covariance matrix, which is roughly on the order of d^3, where d is the dimension of w. A practical point of concern is that matrix inversion is numerically instable, in particular, for matrices that are close to singular, which can lead to serious roundoff errors.

Algorithm 1 Assumed density filtering

$Q_0(w) = \Psi_0(w)$
for $i = 1$ to n **do**
  $Q_{0:i}(w) = \text{Project\_to\_Gaussian}(Q_{0:i-1}(w)\Psi_i(w))$
end for

Algorithm 2 Expectation propagation

$\tilde{\Psi}_0(w) = \Psi_0(w)$
for $i = 1$ to n **do**
  $\tilde{\Psi}_i(w) = 1$
end for
$Q(w) = \prod_{i=0}^{n} \tilde{\Psi}_i(w)$
while not converged **do**
  for $i = 1$ to n **do**
    $Q_{-i}(w) = \frac{Q(w)}{\tilde{\Psi}_i(w)}$
    $Q(w) = \text{Project\_to\_Gaussian}(Q_{-i}(w)\Psi_i(w))$
    $\tilde{\Psi}_i(w) = \frac{Q(w)}{Q_{-i}(w)}$
  end for
end while

Convergence Issues

Sadly enough, expectation propagation is not guaranteed to converge to a fixed point. If it does, this fixed point can be shown to correspond to an extremum of the so-called Bethe free energy, an approximation of the "evidence" $\log P(D)$, under particular consistency and normalization constraints (Minka 2001; Herbrich and Graepel 2006; Heskes and Zoeter 2002; Heskes et al. 2005). These constraints relate to the projection step in Algorithm 2: after convergence, the moments of $Q(w)$ should be equal to the moments of the distribution obtained by taking out a term approximation and putting back the corresponding exact term. This should hold for all i.i.d. observations $i = 1, \ldots, n$ in the factor graph of Fig. 1: so we conclude that, after convergence, the moments ("expectations") of all distributions constructed in this way should be the same. Expectation consistent approximations are based on the exact same idea and indeed turn out to be equivalent to expectation propagation (Heskes et al. 2005).

When expectation propagation does not converge, we can try "damping": instead of replacing the old term approximation by the new one, we replace it by a logconvex combination of the old and the new one. In many cases, damping with a step size 0.1 makes expectation propagation converge, at the expense of requiring more iterations. However, even damping with an infinitesimally small step size is not guaranteed to lead to convergence. In those cases, we can try to minimize the Bethe free energy more explicitly with a so-called double-loop algorithm (Heskes and Zoeter 2002): in the outer loop we compute a convex bound on the Bethe free energy, which we then minimize in the inner loop with an algorithm very similar to standard expectation propagation. Double-loop algorithms are an order of magnitude slower than standard expectation propagation. Recent approaches such as Seeger and Nickisch (2010) provide guaranteed convergence at a much faster rate, but only for specific models.

Generalizations

The running example above serves to illustrate the main idea, but is of course rather restrictive. Expectation propagation can be applied with any member of the exponential family as approximating distribution (Minka 2001; Seeger 2008). The crucial operations are the projection step and the transformation from moment to canonical form: if these can be performed efficiently and robustly, expectation propagation is into play.

In many interesting cases, the model to be learned (here represented as a single variable w) contains a lot of structure. This structure can be exploited by expectation propagation to make it more efficient. For example, when a term only contains a subset of the elements of w, so does its term approximation. Also, we might take as the approximating distribution a distribution that factorizes over the elements of w, instead of a "full" distribution coupling all elements. For a Gaussian, this would amount to a diagonal instead of a full covariance matrix. Such a factorization will lead to lower memory requirements and faster computation, perhaps at the expense of reduced accuracy. More advanced approximations include Tree-EP, where the approximating

structure is a tree, and generalized expectation propagation, which generalizes expectation propagation to include higher-order interactions in the same way as generalized belief propagation generalizes loopy belief propagation (Welling et al. 2005). Systematic higher-order corrections on top of standard expectation propagation lead to improved approximate inference in Gaussian latent variable models (Cseke and Heskes 2011; Opper et al. 2013).

Power expectation propagation (Minka 2005) generalizes expectation propagation by considering a different distance measure in the projection step. Instead of taking the Kullback-Leibler divergence, we can take any so-called α-divergence. $\alpha = 1$ corresponds to the Kullback-Leibler divergence and $\alpha = -1$ to the Kullback-Leibler divergence with the two probabilities interchanged. In the latter case, we obtain a variational method called variational Bayes.

Programs and Data

Code for expectation propagation applied to Gaussian process classification can be found at http://www.gaussianprocess.org/gpml/code/matlab/doc/ or http://becs.aalto.fi/en/research/bayes/gpstuff/. Kevin Murphy's Bayes Net toolbox (https://code.google.com/p/bnt/) can provide a good starting point to write your own code for expectation propagation. Expectation propagation is one of the approximate inference methods implemented in Infer.NET, Microsoft's framework for running Bayesian inference in graphical models (http://research.microsoft.com/en-us/um/cambridge/projects/infernet/).

Applications

Expectation propagation has been applied for, among others, Gaussian process classification (Csato et al. 2002), inference in Bayesian networks and Markov random fields, text classification with Dirichlet models and processes (Minka and Lafferty 2002), logistic regression models for rating players (Herbrich and Graepel 2006), and inference and learning in hybrid and nonlinear dynamic Bayesian networks (Heskes and Zoeter 2002).

Future Directions

From an application point of view, expectation propagation will probably become one of the standard techniques for approximate Bayesian machine learning, much like the Laplace approximation and Monte Carlo methods. Future research may involve questions like

- When does expectation propagation converge? Can we design variants that are guaranteed to converge for any type of model?
- What "power" to use in power expectation propagation for what kind of purposes?
- Can we adapt expectation propagation to handle approximating distributions that are not part of the exponential family? Recent progress in this direction includes Barthelme and Chopin (2014).

Cross-References

▸ Gaussian Process

Recommended Reading

Barthelmé S, Chopin N (2014) Expectation propagation for likelihood-free inference. J Am Stat Assoc 109(505):315–333

Csató L (2002) Gaussian processes – iterative sparse approximations. Ph.D. thesis, Aston University

Cseke B, Heskes TT (2011) Approximate marginals in latent Gaussian models. J Mach Learn Res 12:417–454

Herbrich R, Graepel T (2006) TrueSkill: a Bayesian skill rating system. Technical report (MSR-TR-2006-80), Microsoft Research, Cambridge

Heskes T, Zoeter O (2002) Expectation propagation for approximate inference in dynamic Bayesian networks. In: Darwiche A, Friedman N (eds) Proceedings of the 18th conference on uncertainty in artificial intelligence, Alberta, pp 216–223

Heskes T, Opper M, Wiegerinck W, Winther O, Zoeter O (2005) Approximate inference with expectation constraints. J Stat Mech Theory Exp P11015
Minka T (2001) A family of algorithms for approximate Bayesian inference. Ph.D. thesis, MIT
Minka T (2005) Divergence measures and message passing. Technical report (MSR-TR-2005-173), Microsoft Research, Cambridge
Minka T, Lafferty J (2002) Expectation-propagation for the generative aspect model. In: Darwiche A, Friedman N (eds) Proceedings of the 18th conference on uncertainty in artificial intelligence, Alberta, pp 352–359
Opper M, Paquet U, Winther O (2013) Perturbative corrections for approximate inference in Gaussian latent variable models. J Mach Learn Res 14(1):2857–2898
Opper M, Winther O (2001) Tractable approximations for probabilistic models: the adaptive Thouless-Anderson-Palmer mean field approach. Phys Rev Lett 86:3695–3699
Seeger M (2008) Bayesian inference and optimal design for the sparse linear model. J Mach Learn Res 9: 759–813
Seeger M, Nickisch H (2010) Fast convergent algorithms for expectation propagation approximate Bayesian inference. arXiv preprint arXiv: 1012.3584
Welling M, Minka T, Teh Y (2005) Structured region graphs: morphing EP into GBP. In: Bacchus F, Jaakkola T (eds) Proceedings of the 21st conference on uncertainty in artificial intelligence (UAI), Edinburgh, pp 609

Experience Curve

▸ Learning Curves in Machine Learning

Experience-Based Reasoning

▸ Case-Based Reasoning

Explanation

In ▸ Minimum Message Length, an *explanation is* a code with two parts, where the first part is an *assertion* code and the second part is a *detail* code.

Explanation-Based Generalization for Planning

▸ Explanation-Based Learning for Planning

Explanation-Based Learning

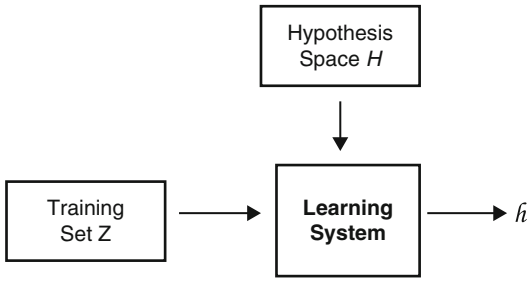
Gerald DeJong[1] and Shiau Hong Lim[2]
[1]University of Illinois at Urbana, Urbana, IL, USA
[2]University of Illinois, Champaign, IL, USA

Synonyms

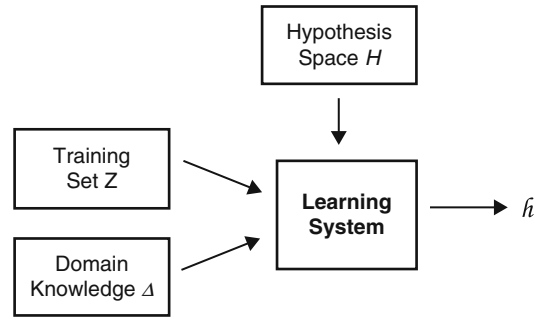
Analytical learning; Deductive learning; EBL; Utility problem

Definition

Explanation-based learning (EBL) is a principled method for exploiting available domain knowledge to improve ▸ supervised learning. Improvement can be in speed of learning, confidence of learning, accuracy of the learned concept, or a combination of these. In modern EBL the domain theory represents an expert's approximate knowledge of complex systematic world behavior. It may be imperfect and incomplete. Inference over the domain knowledge provides *analytic* evidence that compliments the empirical evidence of the training data. By contrast, in original EBL, the domain theory is required to be much stronger; inferred properties are guaranteed. Another important aspect of modern EBL is the interaction between domain knowledge and labeled training examples afforded by explanations. Interaction allows the nonlinear combination of evidence so that the resulting information about the target concept can be much greater than the sum of the information from each evidence source taken independently.



Explanation-Based Learning, Fig. 1 Conventional learner



Explanation-Based Learning, Fig. 2 EBL learner

Motivation and Background

A conventional machine learning system is illustrated in Fig. 1. A hypothesis $\hat{h}$ is selected from a space of candidates H using a training set of labeled examples Z as evidence. It is common to assume that the examples are drawn from some space of well-formed inputs X according to some fixed but unknown distribution $\mathcal{D}$. The quality of $\hat{h}$ is to be judged against different examples similarly selected and labeled. The correct label for an example is specified by some ideal *target concept*, c^*. This is typically some complex world process whose outcome is of interest. The target concept, c^*, will generally not be a member of space of acceptable candidates, H. Rather, the learner tries to find some $\hat{h}$ which is acceptably similar to c^* over $X_{\mathcal{D}}$ and can serve as a computationally tractable stand-in.

Of course, good performance of $\hat{h}$ on Z (its training performance) alone is insufficient. The learner must achieve some statistical guarantee of good performance on the underlying distribution (test performance). If H is too rich and diverse or if Z is too impoverished, a learner is likely to ▸ overfit the data; it may find a pattern in the training data that does not hold in the underlying distribution $X_{\mathcal{D}}$. Test performance will be poor despite good training performance.

An explanation-based learner employs its domain theory, Δ (Fig. 2), as an additional source of information. This domain theory must not be confused with ▸ learning bias, which is present in all learners. Determinations (Russell and Grosof 1987) provide an extreme illustration. These are logical expressions that make strong claims about the world but only after seeing a training example. EBL domain theories are used only to explain. An inferred expression is not guaranteed to hold but only provides analytic evidence.

An explanation for some $z \in Z$ is immediately and easily generalized: The structure of the explanation accounts for why zs assigned classification label should follow from its features. All other examples that meet these conditions are assigned the same classification by the generalized explanation for the same reasons.

Early approaches to EBL (e.g., DeJong and Mooney 1986; Mitchell et al. 1986; Mitchell 1997; Russell and Norvig 2003) were undone by two difficult problems: (1) unavoidable imperfections in the domain theory and (2) the utility problem. The former stems from assuming a conventional semantics for the domain theory. It results in a brittleness and an under-reliance on the training data. Modern EBL is largely a reaction to this difficulty. The utility problem is a consequence of an ill-defined hypothesis space and, as will be discussed later, can be avoided in a straightforward manner.

Structure of Learning System

Explanations and Their Generalization

An *explanation* for a training example is any causal structure, derivable from Δ, which justifies why this training example might merit its teacher-assigned classification label. A *generalized explanation* is the structure of an explanation with-

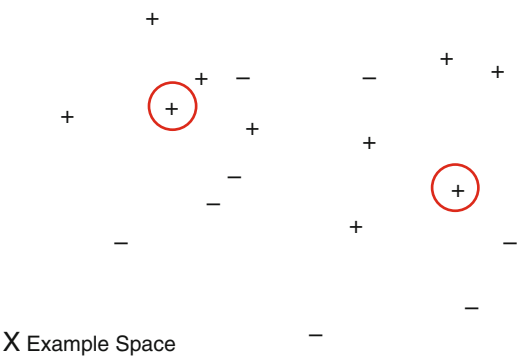
out the commitment to any particular example. The explanation and generalization processes are relatively straightforward and not significantly different from the original EBL algorithms.

The weakness of early EBL is in viewing the components of Δ as constraints. This leads to a view of explanations and their generalizations as *proofs*. Real-world brittleness due to the qualification problem (McCarthy 1980) follows inevitably. In modern EBL, Δ is seen as approximating the underlying world constraints (DeJong 2006; Kimmig et al. 2007). The domain theory is fundamentally a statistical device. Its analytic evidence and the empirical evidence of the training examples both provide a bridge to the real world.

The domain theory introduces new predicates and specifies their significant potential interactions. From a statistical point of view, these are named latent (hidden) features together with a kind of grammar for constructing alternative estimators for them. In short, the domain theory compactly specifies a large set of conceptual structures that an expert believes may be useful in making sense of the domain. If the expert is correct, then patterns of interest will become computationally much more accessible via analytic inference.

One flexible and useful form of a domain theory is sound inference over a set of first-order symbolic logic sentences. In such domain theories, the explanation mechanism can be identical to logical deduction although using a paraconsistent inference mechanism; inference must be well behaved despite inconsistencies in the theory. Generalized explanations are simply "theorems" of Δ that relate a classification label to the values of observable example features. But since the sentences of the theory only approximate world constraints, derivation alone, even via sound inference, is not sufficient evidence to believe a conclusion. Thus, a generalized explanation is only a conjecture. Additional training examples beyond those used to generate each explanation help to estimate the utility of these generalizations.

But analytic mechanisms need not be limited to symbolic logic-like inference. For example, one EBL approach is to distinguish handwritten Chinese characters (Lim et al. 2007) employing a Hough transform as a component of the domain theory. There, an explanation conjectures (hidden) glyph "strokes" to explain how the observed pixels of the training images may realize the image's character label.

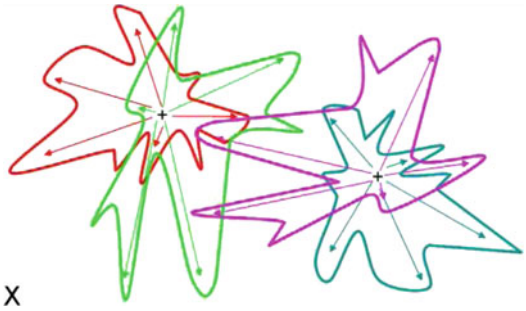


Explanation-Based Learning, Fig. 3 An example space with two designated positive training items

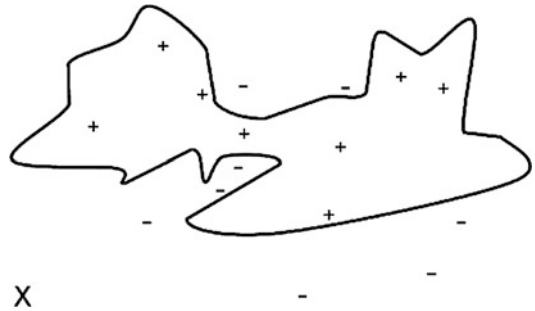
Whatever the form of the analytic inferential mechanism, multiple, quite incompatible explanations can be generated; the same training label can be explained using different input features and postulating different interactions. Such explanations will generalize to cover quite different subsets of X. Figure 3 shows a small training set with two positive examples highlighted. While the explanation process can be applied to all examples both positive and negative, these two will be used to illustrate. In this illustration, just two explanations are constructed for each of the highlighted training examples. Figure 4 shows the generalized extensions of these four explanations in the example space. The region enclosed by each contour is meant to denote the subset of X conjectured to merit the same classification as the explained example. Explanations make no claim about the labels for examples outside their extension.

Evaluation and Hypothesis Selection

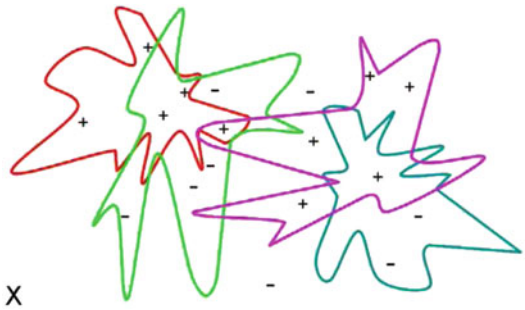
Additional training examples that fall within the extension of a generalized explanation help to evaluate it empirically. This is shown in Fig. 5. The estimated utility of a generalized explanation



Explanation-Based Learning, Fig. 4 Four constructed explanations are sufficient to cover the positive examples



Explanation-Based Learning, Fig. 6 An element from H that approximates the weighted explanations



Explanation-Based Learning, Fig. 5 Explanations are evaluated with other training examples

reflects (1) the generalized explanation's empirical accuracy on these training examples, (2) the inferential effort required to derive the explanation (see DeJong 2006), and (3) the redundancies and interactions with other generalized explanations (higher utility is estimated if its correct predictions are less commonly shared by other generalized explanations).

The estimated utilities define an EBL classifier as a mixture of the generalized explanations each weighted by its estimated utility:

$$\hat{c}_{\mathrm{EBL}}(x) = \sum_{g \in \mathrm{GE}(Z,\Delta)} u_g \cdot g(x),$$

where $\mathrm{GE}(Z, \Delta)$ denotes the generalized explanations for Z from Δ and u_g is the estimated utility for g. This corresponds to a voting scheme where each generalized explanation that claims to apply to an example casts a vote in proportion to its estimated utility. The votes are normalized over the utilities of voting generalized explanations. The mixture scheme is similar to that of sleeping experts (Freund et al. 1997). This EBL classifier approximates the target concept c^*. But unlike the approximation chosen by a conventional learner, $\hat{c}_{\mathrm{EBL}}$ reflects the information of Δ in addition to Z.

The final step is to select a hypothesis $\hat{h}$ from H. The EBL concept $\hat{c}_{\mathrm{EBL}}$ is used to guide this choice. Figure 6 illustrates the selection of a $\hat{h} \in H$, which is a good approximation to a utility-blended mixture of Fig. 5. This final step, selecting a hypothesis from H, is important but was omitted in original EBL. These systems employed generalized explanations directly. Unfortunately, such classifiers suffer from a difficulty known as the *utility problem* (Minton 1990). Note this is a slightly different use of the term *utility*, referring to the performance of an application system. This system can be harmed more than helped by concepts such as $\hat{c}_{\mathrm{EBL}}$, even if these concepts provide highly accurate classification. Essentially, the average cost of evaluating an EBL concept may outweigh the average benefit that it provides to the application system. It is now clear that this utility problem is simply the manifestation of a poorly structured hypothesis space. Note that, in general, an EBL classifier itself will not be an element of the space of acceptable hypotheses H. Previous approaches to the utility problem (Minton 1990; Gratch and DeJong 1992; Greiner and Jurisica 1992; Etzioni 1993) identify and disallow offending EBL concepts. However, the root cause is addressed by employing the EBL concept as a guidance in selecting a $\hat{h} \in H$

rather than using $\hat{c}_{\text{EBL}}$ directly. Without this last step, H is completely ignored. But H embodies all of the information in the learning problem about what makes an acceptable hypothesis. The "utility problem" is simply the manifestation of leaving out this important information.

Literature

The roots and motivation for EBL extend at least to the MACROPs of STRIPS (Fikes et al. 1972). The importance of explanations of training examples was first suggested in DeJong (1981). The standard references for the early EBL work are Mitchell et al. (1986) and DeJong and Mooney (1986). When covering EBL, current textbooks give somewhat refined versions of this early approach (Mitchell 1997; Russell and Norvig 2003). Important related ideas include determinations (Russell and Grosof 1987), chunking (Laird et al. 1986), and knowledge compilation (Anderson 1986). EBL's ability to employ first-order theories make it an attractive compliment to learning Horn theories with ▸ Inductive Logic Programming (Hirsh 1987; Bruynooghe et al. 1989; Pazzani and Kibler 1992; Zelle and Mooney 1993). The problem of imperfect domain theories was recognized early, and there have been many approaches (Flann and Dietterich 1989; Genest et al. 1990; Towell et al. 1991; Cohen 1992; Thrun and Mitchell 1993; Ourston and Mooney 1994). But with modern statistical learning ascending to the dominant paradigm of the field, interest in analytic approaches waned. The current resurgence of interest is largely driven by placing EBL in a modern statistically sophisticated framework that nonetheless is still able to exploit a first-order expressiveness (DeJong 2006; Kimmig et al. 2007; Lim et al. 2007; Sun and DeJong 2005).

Cross-References

- ▸ Deductive Learning
- ▸ Explanation-Based Learning for Planning
- ▸ Speedup Learning

Recommended Reading

Anderson J (1986) Knowledge compilation: the general learning mechanism. In: Michalski R, Carbonell J, Mitchell T (eds) Machine learning II. Morgan Kaufmann, San Mateo, pp 289–310

Bruynooghe M, De Raedt L, De Schreye D (1989) Explanation based program transformation. In: IJCAI'89: proceedings of the eleventh international joint conference on artificial intelligence, Detroit, pp 407–412

Cohen WW (1992) Abductive explanation-based learning: a solution to the multiple inconsistent explanation problem. Mach Learn 8:167–219

DeJong G (1981) Generalizations based on explanations. In: IJCAI'81: proceedings of the seventh international joint conference on artificial intelligence, Vancouver, pp 67–69

DeJong G (2006) Toward robust real-world inference: a new perspective on explanation-based learning. In: ECML06: proceedings of the seventeenth European conference on machine learning, Berlin. Springer, Heidelberg, pp 102–113

DeJong G, Mooney R (1986) Explanation-based learning: an alternative view. Mach Learn 1(2):145–176

Etzioni O (1993) A structural theory of explanation-based learning. Artif Intell 60(1):93–139

Fikes R, Hart PE, Nilsson NJ (1972) Learning and executing generalized robot plans. Artif Intell 3(1–3):251–288

Flann NS, Dietterich TG (1989) A study of explanation-based methods for inductive learning. Mach Learn 4:187–226

Freund Y, Schapire RE, Singer Y, Warmuth MK (1997) Using and combining predictors that specialize. In: Twenty-ninth annual ACM symposium on the theory of computing, El Paso, pp 334–343

Genest J, Matwin S, Plante B (1990) Explanation-based learning with incomplete theories: a three-step approach. In: Proceedings of the seventh international conference on machine learning, Austin, pp 286–294

Gratch J, DeJong G (1992) Composer: a probabilistic solution to the utility problem in speed-up learning. In: AAAI, San Jose, pp 235–240

Greiner R, Jurisica I (1992) A statistical approach to solving the EBL utility problem. In: National conference on artificial intelligence, San Jose, pp 241–248

Hirsh H (1987) Explanation-based generalization in a logic-programming environment. In: IJCAI'87: proceedings of the tenth international joint conference on artificial intelligence, Milan, pp 221–227

Kimmig A, De Raedt L, Toivonen H (2007) Probabilistic explanation based learning. In: ECML'07: proceedings of the eighteenth European conference on machine learning, Warsaw, pp 176–187

Laird JE, Rosenbloom PS, Newell A (1986) Chunking in soar: the anatomy of a general learning mechanism. Mach Learn 1(1):11–46

Lim SH, Wang L-L, DeJong G (2007) Explanation-based feature construction. In: IJCAI'07: proceedings of the twentieth international joint conference on artificial intelligence, Hyderabad, pp 931–936
McCarthy J (1980) Circumscription – a form of non-monotonic reasoning. Artif Intell 13:27–39
Minton S (1990) Quantitative results concerning the utility of explanation-based learning. Artif Intell 42(2–3):363–391
Mitchell T (1997) Machine learning. McGraw-Hill, New York
Mitchell T, Keller R, Kedar-Cabelli S (1986) Explanation-based generalization: a unifying view. Mach Learn 1(1):47–80
Ourston D, Mooney RJ (1994) Theory refinement combining analytical and empirical methods. Artif Intell 66(2):273–309
Pazzani MJ, Kibler DF (1992) The utility of knowledge in inductive learning. Mach Learn 9:57–94
Russell SJ, Grosof BN (1987) A declarative approach to bias in concept learning. In: AAAI, Seattle, pp 505–510
Russell S, Norvig P (2003) Artificial intelligence: a modern approach, 2nd edn. Prentice-Hall, Englewood Cliffs
Sun Q, DeJong G (2005) Feature kernel functions: improving SVMs using high-level knowledge. In: CVPR (2), San Diego, pp 177–183
Thrun S, Mitchell TM (1993) Integrating inductive neural network learning and explanation-based learning. In: IJCAI'93: proceedings of the thirteenth international joint conference on artificial intelligence, Chambery, pp 930–936
Towell GG, Craven M, Shavlik JW (1991) Constructive induction in knowledge-based neural networks. In: proceedings of the eighth international conference on machine learning, Evanston, pp 213–217
Zelle JM, Mooney RJ (1993) Combining Foil and EBG to speed-up logic programs. In: IJCAI'93: proceedings of the thirteenth international joint conference on artificial intelligence, Chambery, pp 1106–1113

Explanation-Based Learning for Planning

Subbarao Kambhampati[1] and Sungwook Yoon[2]
[1]Arizona State University, Tempe, AZ, USA
[2]MapR, San Jose, CA, USA

Synonyms

Explanation-based generalization for planning; Speedup learning for planning

Definition

▸ Explanation-based learning (EBL) involves using prior knowledge to explain ("prove") why the training example has the label it is given and using this explanation to guide the learning. Since the explanations are often able to pinpoint the features of the example that justify its label, EBL techniques are able to get by with much fewer number of training examples. On the flip side, unlike general classification learners, EBL requires prior knowledge (aka "domain theory/model") in addition to labeled training examples – a requirement that is not easily met in some scenarios. Since many planning and problem-solving agents do start with declarative domain theories (consisting at least of descriptions of actions along with their preconditions and effects), EBL has been a popular learning technique for planning.

Dimensions of Variation

The application of EBL in planning varies along several dimensions: whether the learning was for improving the speed and quality of the underlying planner (Etzioni 1993; Kambhampati 1994; Kambhampati et al. 1996; Minton et al. 1989; Yoon et al. 2008) or acquire the domain model (Levine and DeJong 2006), whether it was done from successes (Kambhampati 1994; Yoon et al. 2008) or failures (Minton et al. 1989; Ihrig and Kambhampati 1997), whether the explanations were based on complete/correct (Minton et al. 1989; Kambhampati et al. 1996) or partial domain theories (Yoon et al.), whether learning is based on single (Kambhampati 1994; Kambhampati et al. 1996; Minton et al. 1989) or multiple examples (Flann and Dietterich 1989; Estlin and Mooney 1997) (where, in the latter case, inductive learning is used in conjunction with EBL), and finally whether the planner whose performance EBL aims to improve is a means-ends analysis one (Minton et al. 1989), partial-order planner (Estlin and Mooney 1997), or a heuristic search planner (Yoon et al.).

EBL techniques have been used in planning both to improve search and to reduce domain

modeling burden (although the former has received more attention by far). In the former case, EBL is used to learn "control knowledge" to speed up the search process (Minton et al. 1989; Kambhampati et al. 1996) or to improve the quality of the solutions found by the search process (Estlin and Mooney 1997). In the latter case, EBL is used to develop domain models (e.g., action models) (Levine and DeJong 2006).

EBL for search improvement involves either remembering and reusing successful plans or learning search control rules to avoid failing search branches. Other variations include learning effective indexing of stored cases from retrieval failures (Ihrig and Kambhampati 1997) and learning "adjustments" to the default heuristic used by the underlying search.

Another important issue is the degree of completeness/correctness of the underlying background theory used to explain examples. If the theory is complete and correct, then learning is possible from a single example. This type of EBL has been called "analytical learning." When the theory is partial, EBL still is effective in narrowing down the set of potentially relevant features of the training example. These features can then be used within an inductive learner. Within planning, EBL has been used in the context of complete/correct as well as partial domain models.

A final dimension of variation that differentiated a large number of research efforts is the type of underlying planner. Initially, EBL was used on top of means-ends analysis planners (cf. PRODIGY, Minton et al. 1989). Later work focused on partial-order planners (e.g., Kambhampati et al. 1996; Estlin and Mooney 1997). More recently, the focus has been on forward search state-space planners (Yoon et al. 2008).

Learning from Success: Explanation-Based Generalization

When learning from successful cases (plans), the training examples comprise of successful plans, and the explanations involve proofs showing that the plan, as it is given, is able to support the goals. Only the parts of the plan that take part in this proof are relevant for justifying the success of the plan. The plan is thus "generalized" by removing extraneous actions that do not take part in the proof. Object identifiers and action orderings are also generalized as long as the generalization does not affect the proof of correctness (Kambhampati 1994). The output of the learning phase is thus a variablized plan containing a subset of the constraints (actions, orderings, object identity constraints) of the original plan. This is then typically indexed and used as a macro-operator to speed up later search.

For example, given a planning problem of starting with an initial state where five blocks, A, B, C, D, and E, are on the table, and the problem requires that in the goal state A must be on B and C must be on D and a plan P that is a sequence of actions *pickup A, stack A on B, pickup E, putdown E, Pickup C, stack C on D*, the explanation-based learner might output the generalization *do in any order* {*pickup x, stack x on y*} {*pick up z, stack z on w*} for the generalized goals *on* (x, y) *and on* (w, z), starting from a state where x, y, z, and w are all on the table and clear, and each of them denotes a distinct block.

One general class of such proof schema involves showing that every top-level goal of the planning problem as well as the precondition of every action is established and protected. Establishment requires that there is an action in the plan that gives that condition, and protection requires that once established, the condition is not deleted by any intervening action.

A crucial point is that the extent of generalization depends on the flexibility of the proof strategy used. Kambhampati and Kedar (1994) discuss a spectrum of generalization strategies associated with a spectrum of proof strategies, while Shavlik (1990) discusses how the number of actions in the plan can also be generalized.

Learning from Failure

When learning from the failure of a search branch, EBL starts by analyzing the plans at

the failing nodes and constructing an explanation of failure. The failure explanation is just a subset of constraints in the plan at the current search node, which, in conjunction with domain theory, ensures that no successful solution can be reached by further refining this plan. The explanations can range from direct constraint inconsistencies (e.g., ordering cycles) to indirect violation of domain axioms (e.g., the plan requiring both clear(B) and On(A,B) to be satisfied at the same time point). The explanations at the leaf nodes are "regressed" over the decisions in the search tree to higher-level nodes to get explanations of (implicit) failures in these higher-level nodes. The search control rules can then essentially recommend pruning any search node which satisfies a failure explanation.

The deep affinity between EBL from search failures and the idea of ▸ nogood learning and dependency-directed backtracking in CSP is explored in Kambhampati (1998). As in dependency-directed backtracking, the more succinct the explanation, the higher the chance of learning effective control rules. Note that effectiveness here is defined in terms of the match costs involved in checking whether the rule is applicable and the search reductions provided when it is applicable. Significant work has been done to identify classes of failure explanation that are expected to lead to ineffective rules (Etzioni 1993). In contrast to CSP that has a finite depth search tree, one challenge in planning is that often an unpromising search node might not exhibit any direct failure with a succinct explanation and is abandoned by the search for heuristic reasons (such as the fact that the node crosses a depth limit threshold). Strategies for finding implicit explanations of failure (using domain axioms), as well as getting by with incomplete explanations of failure, are discussed in Kambhampati et al. (1996). EBL from failures has also been applied to retrieval (rather than search) failures. In this case, the failure of extending a plan retrieved from the library to solve a new problem is used to learn new indexing schemes that inhibit that case from being retrieved in such situations (Ihrig and Kambhampati 1997).

Learning Adjustments to Heuristics

Most recent work in planning has been in the context of heuristic search planners, where learning from failures does not work as well (since the heuristic search may change directions much before a given search branch ends in an explainable failure). One way of helping such planners is to improve their default heuristic (Yoon et al. 2008). Given a heuristic $h(s)$ that gives the heuristic estimate of state s, the aim in Yoon et al. is to learn an adjustment $\delta(s)$ that is added to $h(s)$ to get a better estimate of $h^*(s)$ – the true cost of state s. The system has access to actual plan traces (which can be obtained by having the underlying planner solve some problems from scratch). For each state s on the trace, we know the true distance of state s from the goal state, and we can also compute the $h(s)$ value with respect to the default heuristic. This gives the learner a set of training examples which are pairs of states and the adjustments they needed to make the default heuristic meet the true distance. In order to learn the $\delta(s)$ from this training data, we need to enumerate the features of state s that are relevant to it needing the specific adjustment. This is where EBL comes in. Specifically, one way of enumerating the relevant features is to explain why s has the default heuristic value. This, in turn, is done by taking the features of the relaxed plan for state s. Since the relaxed plan is a plan that assumes away all negative interactions between the actions, relaxed plan features can be seen as features of the explanation of the label for state s in terms of a *partial domain theory* (one which ignores all the deletes of all actions).

EBL from Incomplete Domain Theories

While most early efforts for speedup focused on complete and correct theories, several efforts also looked at speedup learning from incomplete theories. The so-called Lazy EBL approaches (Tadepalli 1989; Chien 1989) work by first constructing partial explanations and subsequently refining the over-general rules learned. Other ap-

proaches that use similar ideas outside planning include Flann and Dietterich (1989) and Cohen (1992). As we noted above, the work by Yoon et al. (2008) can also be seen as basing learning (in their case of adjustments to a default heuristic function) w.r.t. a partial domain theory.

EBL to Learn Domain Knowledge

Although most work in EBL for planning has been focused on speedup, there has also been some work aimed at learning domain knowledge (rather than control knowledge). Of particular interest is "operationalizing" a complex, if opaque, domain model by learning from it a simplified domain model that is adequate to efficiently solve an expected distribution of problems. The recent work by Levine and DeJong (2006) is an example of such an effort.

EBL and Knowledge-Level Learning

Although the focus of this article is on EBL as applied to planning, we need to foreground one general issue: whether EBL is capable of knowledge-level learning or not. A popular misconception of EBL is that since it depends on a complete and correct domain theory, no knowledge-level learning is possible, and speedup learning is the only possibility. (The origins of this misconception can be traced back to the very beginning. The two seminal articles on EBL in the very first issue of the Machine Learning journal differed profoundly in their interpretations of EBL. While Mitchell et al. (1986) assumed that EBL by default works with complete and correct theories (thus precluding any knowledge-level learning), Levine and DeJong (2006) provides a more general view of EBL that uses background knowledge – whether or not it is complete – to focus the generalization (and as such can be seen as a knowledge-based feature-selection step for a subsequent inductive learner).) As we noted at the outset however, EBL is not required to depend on complete and correct domain theories, and when it does not, knowledge-level learning is indeed possible.

Utility Problem and Its Nonexclusive Relation to EBL

As we saw above, much early work in EBL for planning focused on speedup for the underlying planner. Some of the knowledge learned for speedup – especially control rules and macro-operators – can also adversely affect the search by increasing either the search space size (macros) or per-node cost (matching control rules). Clearly, in order for the net effect to be positive, care needs to be exercised as to which control rules and/or macros are stored. This has been called the "utility problem" (Minton 1990), and significant attention has been paid to develop strategies that either dynamically evaluate the utility of the learned control knowledge (and forget useless rules) (Markovitch and Scott 1988; Minton 1990) or select the set of rules that best serve a given distribution of problem instances (Gratch et al. 1994).

Despite the prominent attention given to the utility problem, it is important to note the nonexclusive connection between EBL and utility problem. We note that *any* strategy that aims to provide/acquire control knowledge will suffer from the utility problem. For example, utility problem also holds for inductive learning techniques that were used to learn control knowledge (cf. Leckie and Zukerman 1993). In other words, it is not special to EBL but rather to the specific application task. We note that it is both possible to do speedup learning that is less susceptible to the utility problem (e.g., learn adjustments to heuristics, Yoon et al. 2008) and possible to use EBL for knowledge-level learning (Levine and DeJong 2006).

Current Status

EBL for planning was very much in vogue in the late 1980s and early 1990s. However, as the speed of the underlying planners increased drastically,

the need for learning as a crutch to improve search efficiency reduced. There has however been a recent resurgence of interest, both in further speeding up the planners and in learning domain models. Starting 2008, there is a new track in the International Planning Competition devoted to learning methods for planning. In the first year, the emphasis was on speedup learning. ObtuseWedge, a system that uses EBL analysis to learn adjustments to the default heuristic, was among the winners of the track. The DARPA integrated learning initiative, and interest in model-lite planning have also brought focus back to EBL for planning – this time with partial domain theories.

Additional Reading

The tutorial (Yoon and Kambhampati 2007) provides an up-to-date and broader overview of learning techniques applied to planning and contains significant discussion of EBL techniques. The paper Zimmerman and Kambhampati (2003) provides a survey of machine learning techniques used in planning and includes a more comprehensive listing of research efforts that applied EBL in planning.

Cross-References

- ▸ Explanation-Based Learning
- ▸ Speedup Learning

Recommanded Reading

Bhatnagar N, Mostow J (1994) On-line learning from search failures. Mach Learn 15(1):69–117

Borrajo D, Veloso MM (1997) Lazy incremental learning of control knowledge for efficiently obtaining quality plans. Artif Intell Rev 11(1–5):371–405

Chien SA (1989) Using and refining simplifications: explanation-based learning of plans in intractable domains. In: IJCAI 1989, Detroit, pp 590–595

Cohen WW (1992) Abductive explanation-based learning: a solution to the multiple inconsistent explanation problem. Mach Learn 8:167–219

DeJong G, Mooney RJ (1986) Explanation-based learning: an alternative view. Mach Learn 1(2):145–176

Estlin TA, Mooney RJ (1997) Learning to improve both efficiency and quality of planning. In: IJCAI 1997, Nagoya, pp 1227–1233

Etzioni O (1993) A structural theory of explanation-based learning. Artif Intell 60(1):93–139

Flann NS, Dietterich TG (1989) A study of explanation-based methods for inductive learning. Mach Learn 4:187–226

Gratch J, Chien SA, DeJong G (1994) Improving learning performance through rational resource allocation. In: AAAI 1994, Seattle, pp 576–581

Ihrig LH, Kambhampati S (1997) Storing and indexing plan derivations through explanation-based analysis of retrieval failures. J Artif Intell Res 7:161–198

Kambhampati S (1994) A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. Artif Intell 67(1):29–70

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

Kambhampati S, Katukam S, Qu Y (1996) Failure driven dynamic search control for partial order planners: an explanation based approach. Artif Intell 88(1–2):253–315

Leckie C, Zukerman I (1993) An inductive approach to learning search control rules for planning. In: IJCAI 1993, Chambéry, pp 1100–1105

Levine G, DeJong G (2006) Explanation-based acquisition of planning operators. In: ICAPS 2006, Cumbria, pp 152–161

Markovitch S, Scott PD (1988) The role of forgetting in learning. In: ML 1988, Ann Arbor, pp 459–465

Minton S (1990) Quantitative results concerning the utility of explanation-based learning. Artif Intell 42(2–3):363–391

Minton S, Carbonell JG, Knoblock CA, Kuokka D, Etzioni O, Gil Y (1989) Explanation-based learning: a problem solving perspective. Artif Intell 40(1–3):63–118

Mitchell TM, Keller RM, Kedar-Cabelli ST (1986) Explanation-based generalization: a unifying view. Mach Learn 1(1):47–80

Shavlik JW (1990) Acquiring recursive and iterative concepts with explanation-based learning. Mach Learn 5:39–40

Tadepalli P (1989) Lazy explanation based learning: a solution to the intractable theory problem. In: IJCAI 1989, Detroit, pp 694–700

Yoon S, Fern A, Givan R (2008) Learning control knowledge for forward search planning. J Mach Learn Res 9:683–718

Yoon S, Kambhampati S (2007) Learning for planning. Tutorial delivered at ICAPS 2007. http://rakaposhi.eas.asu.edu/learn-plan.html

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