Machine Learning

(course 395)

Introduction to Machine Learning

&

Case-Based Reasoning

Maja Pantic

Prologue

These notes refer to the course of Machine Learning (course 395), Computing Department, Imperial College London. The goal of this syllabus is to summarize the basics of machine learning and to provide a detailed explanation of case-based reasoning.

Part 1: Introduction to Machine Learning

This chapter introduces the term "machine learning" and defines what do we mean while using this term. This is a very short summary of the work of Mitchell [8].

Part 2: Case-based Reasoning

This chapter discusses Case-Based Reasoning. This is an adaptation of the work of Pantic [10].

1. Introduction to Machine Learning

This chapter is concerned with the term "machine-learning" and defines what do we mean while using this term. It also provides a short overview of a number of well-known learning paradigms. For a more profound discussion about different learning algorithms, theoretical results, and applications, the reader is referred to [8].

1.1 What does the term "machine learning" denote?

Machine learning is inherently a multidisciplinary field. It draws on results from research fields as diverse as:

- *Artificial Intelligence*: AI forms a theoretical and methodological basis for learning symbolic representations of concepts, learning in terms of classification and pattern recognition problems, and learning by using prior knowledge together with training data as a guideline.
- *Bayesian methods*: the Bayes' theorem forms the basis for calculating probabilities of hypotheses, the basis of the naïve Bayes classifier, and the basis of algorithms for estimating values of unobserved variables.
- *Computational complexity theory*: This theory imposes the theoretical bounds on the inherent complexity of different learning tasks measured in terms of computational effort, number of training examples, number of mistakes, etc.
- *Control theory*: This theory forms the theoretical foundation of procedures that learn to control processes in order to optimize predefined objectives and to predict the next state of the process they are controlling.
- *Information theory*: Measures of entropy and optimal codes are germane and central to the issue of delimiting optimal training sequences for encoding a hypothesis.
- *Philosophy*: Philosophical argumentations like "the simplest hypothesis is the best" underlie the reasoning process of machine learning algorithms.
- *Psychology*: The view on human reasoning and problem-solving initiated many machine learning models (e.g., see the discussion on Case-Based Reasoning in chapter 2).
- *Neurobiology*: Information processing found in biological organisms motivated Artificial Neural Network models of learning (see chapter 4 in [8]).

As delimited by the definition given by Mitchell [8], a computer program is said to learn from experience ***E*** with respect to some class of tasks ***T*** and performance measure ***P*** if its performance at tasks in ***T***, as measured by ***P***, improves with experience ***E***. For example, an automated facial expression classifier which classifies facial expressions in terms of user-defined interpretation labels (e.g., [11]), improves its performance *as measured by its ability to accomplish user-defined interpretations* at the class of tasks involving *classification of facial expressions*, through experience *obtained by interacting with the user on the meanings that he/she associates with different facial expressions*. In general, in a well-defined learning problem, these three features must be identified (i.e. the class of tasks ***T***, the measure of performance to be improved ***P***, and the source of experience ***E***). Once the learning problem is defined, the next step in designing a learning system is to delimit exactly:

- the type of knowledge to be learned,

- the representation of this target knowledge (i.e. the definition of target function to be learned, which when utilized will produce for any instance of a new problem as input a trace of its solution as output), and
- the learning mechanism to apply.

Different target knowledge (hypotheses space) representations are appropriate for learning different kinds of target functions. For each of these hypothesis representations, the corresponding learning algorithm takes advantage of a different underlying structure to organize the search through the hypotheses space. Therefore, deciding about the issues listed above involves searching a very large space of alternative approaches to determine the one that best fits the defined learning problem. In order to decide a machine learning algorithm which will perform best for the given problem and the given target function, it is useful to analyze the relationships between the size of the hypotheses space, the completeness of it, the number of training examples available, the prior knowledge held by the learner, and the confidence we can have that a hypothesis that is consistent with the training data will correctly generalize to unseen examples.

Though, generally, learning is considered as one of the basic facets of intelligence, not all AI techniques are capable of learning. Expert systems are an obvious example, at least in their most common form (see chapter 2 in [9]).

1.2 The primary machine learning approaches and algorithms

Decision Trees
Decision tree learning is one of the most widely used and practical methods for inductive inference. It is a method for approximation of discrete-valued functions, in which a tree represents the learned function. A decision tree is in a nutshell a discrete value functional mapping, a classifier. Each node in the decision tree specifies a test of some attribute of the query instance, and each branch descending from that node corresponds to one of the possible values for this attribute. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute. This process is repeated for the sub-tree rooted at the new node as long as it takes to reach the appropriate leaf node, then returning the classification associated with this leaf. Several algorithms are available that can be used to construct a tree based on some data set. A typical example is the ID3 algorithm proposed in [13]. This is a greedy search algorithm that constructs the tree recursively and chooses at each step the attribute to be tested so that the separation of the data examples is optimal. This decision-tree learning method searches a complete hypothesis space (i.e. the space of all possible decision trees) and, thus, avoids difficulties of restricted hypothesis spaces (i.e. that the target function might not be present in the hypothesis space). Its inductive bias is a preference for small trees over large trees. Experiments that compare decision-tree learning and other learning methods can be found in numerous papers, for example, in [5] and [17].

Artificial Neural Networks
Artificial Neural Networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued target functions from examples. Algorithms such as backpropagation use gradient descent to tune network parameters to best fit a training set of input-output pairs. ANN learning is robust to errors in the training data and has been

successfully applied to problems such as interpreting visual scenes, speech recognition, etc. (see chapter 4 in [8]).

Learning set of rules

One of the most expressive and human readable representations of a learned target function is a set of if-then rules that jointly define the function. One way to learn sets of rules is to learn a decision tree first, then translate the tree into an equivalent set of rules; one rule for each leaf node in the tree. A quite successful method for converting the learned tree into a set of rules is a technique called *rule post pruning* used by the C4.5 algorithm [13], which represents an extension of the original ID3 algorithm.

Another way to convert a tree into a set of rules is to apply a *sequential covering algorithm* for learning sets of rules based upon the strategy of learning one rule, removing the data it covers and then iterating this process. To elaborate, given a LSR (learn-single-rule) subroutine, invoke it on all the available training examples, remove any positive examples covered by the rule it learns, then invoke it again to learn a second rule based on the remaining training examples. Thus, a sequential covering algorithm sequentially learns a set of (disjunctive) rules that together cover the full set of positive examples. Because this algorithm carries out a greedy search, so it formulizes a sequence of rules without backtracking, the smallest or best set of rules that cover the training examples is not necessarily found. A prototypical sequential covering algorithm is the *general-to-specific beam search* which searches through the space of possible rules maintaining $\boldsymbol{k}$ best candidates, then generates descendents for each of these $\boldsymbol{k}$ best candidates, and again reduces the resulting set to $\boldsymbol{k}$ most promising members. This algorithm has been used by the CN2 program [4]. Many variations on this approach have been explored [8].

Inductive Logic Programming

The previous subsection discussed algorithms for learning sets of propositional (i.e. variable-free) rules. This subsection is considered with learning rules that contain variables, in particular, learning first-order Horn theories. Inductive learning of first-order rules is also referred to as *Inductive Logic Programming* (ILP), because this process can be viewed as automatically inferring PROLOG[1] programs from examples. A variety of algorithms has been proposed for learning first-order rules. A typical example is FOIL, which is an extension of the sequential covering algorithms to first-order representations.

Another approach to inductive logic programming is *inverse deduction*, which is based upon the simple observation that induction is just the inverse of deduction. In other words, the problem of induction is to find a hypothesis $\boldsymbol{h}$ that satisfies the constraint $(\forall\langle \boldsymbol{x_i}, \boldsymbol{f(x_i)}\rangle \in \boldsymbol{D})$ $(\boldsymbol{B} \wedge \boldsymbol{h} \wedge \boldsymbol{x_i}) \vdash \boldsymbol{f(x_i)}$, where $\boldsymbol{B}$ is general background information, $\boldsymbol{x_1 \ldots x_n}$ are descriptions of the instances in the training data $\boldsymbol{D}$, $\boldsymbol{f(x_1) \ldots f(x_n)}$ are the target values of the training instances, and expression $\boldsymbol{Z} \vdash \boldsymbol{C}$ is read "$\boldsymbol{C}$ follows deductively from $\boldsymbol{Z}$". A prototypical algorithm based upon inverse deduction principle is CIGOL, introduced by Muggleton and Buntine in 1988, which uses the *inverse resolution*, an operator that is the inverse of the *deductive resolution* operator, introduced by Robinson in 1965, and commonly used for mechanical theorem proving.

Instance-Based Learning

In contrast to learning methods that construct a general, explicit description of the target function when training examples are provided, instance-based learning methods simply store

[1] PROLOG is a general purpose, Turing-equivalent programming language in which programs are expressed as collections of Horn clauses.

the training examples. Generalizing beyond these examples is postponed until a new instance must be classified: given a new instance, its relations to the already stored examples are examined in order to assign a target function value (the classification) for the new instance. Due to this property, instance-based learning methods are also called *lazy learning methods*, as opposed to the *eager learning methods* represented by all other learning algorithms discussed in this section. Examples of instance-based learning include nearest-neighbor learning and locally weighted regression methods. Instance-based learning also includes case-based reasoning methods that use more complex, symbolic representations for instances. An overview of the topic can be found in [8]. A survey of methods for locally weighted regression is given in [3]. Chapter 2 of this syllabus provides a detailed discussion on case-based reasoning.
A key advantage of instance-based learning as a delayed, or lazy, learning method is that instead of estimating the target function once for the entire instance space, these methods can estimate it locally and differently for each new instance to be classified. Yet, these methods are at a disadvantage because of their computation and memory/storage requirements.

Genetic Algorithms
Genetic Algorithms (GA) are optimization techniques providing an approach to learning that is based loosely on simulated evolution. One thing that distinguishes GA from other optimization algorithms is that GA simultaneously work on large sets (populations) of possible solutions. The search for an appropriate hypothesis begins with a population of initial hypotheses. Members of the current population give rise to the next generation population by means of operations such as random mutation and crossover, which are patterned after biological evolution processes. At each iteration, the hypotheses in the current population are evaluated relative to a given measure of fitness and the most fit members of the population are selected to produce new offspring that replace the least fit members of the population. To elaborate, the learning task of GA is to find the optimal hypothesis according to the predefined fitness function.
Evolution-based computational approaches have been explored since the early days of computer science. Evolutionary programming as a method for finite-state machine evolution has been developed by Folgel and colleagues in 1966. Genetic algorithms have been first introduced by Holland in 1962. An overview of the subject can be found in [8] and [9]. GA are especially suited to tasks in which hypotheses are complex (e.g. sets of rules for robot control, sets of optimal routes, etc.) and in which the objective to be optimized may be an indirect function of the hypotheses. A variant of GA is *genetic programming*, in which the hypotheses being manipulated are computer programs rather than bit strings[2]. Genetic programming has been demonstrated to learn programs for tasks such as simulated robot control and recognizing objects in visual scenes (including human facial expressions).

Reinforcement Learning
Reinforcement learning addresses the question of how an autonomous agent (see syllabus on Intelligent Agents and Multi Agent Systems), which senses and acts in its environment, can learn to choose optimal actions to accomplish its goals. This generic problem covers learning tasks such as to control CAM tools and robots, to optimize operations in factories, to search Internet, to play board games, etc. In a nutshell, reinforcement learning is reward hunting. Namely, each time a given agent performs an action in its environment, a trainer may provide a reward or penalty to indicate the desirability of the resulting state; the goal of the agent is to learn an action policy that maximizes the total reward it will receive from any starting state.

[2] Though hypotheses may be represented by symbolic expressions or even computer programs, they are usually described by bit strings. The interpretation of these bit strings depends on the actual application.

The reinforcement learning methodology fits a problem setting known as a Markov decision process, in which the outcome of applying any action to any state depends only on this action and this state as opposed to being dependent on preceding actions or states. A prototypical reinforcement learning algorithm is Q-learning, in which the agent learns the evaluation function ***Q(s, a)*** representing the maximum expected, cumulative reward the agent can achieve by applying action ***a*** to state ***s***. Watkins introduced Q learning in 1989 with the goal of acquiring optimal policies when the reward and action transition functions are unknown. Recently a survey of the related methods has been is given by Sutton and Barto [18].

1.3 Vantages and disadvantages of machine learning

The major vantage of a learning system is its ability to adapt to a changing environment. Of course, the existing machine-learning techniques are still far from enabling computers to learn nearly as well as people. Yet algorithms have been invented that are effective for certain types of learning tasks. In the late 90s, a formalized theoretical foundation of learning was established [8], and many practical computer programs have been developed to enable different types of learning. Machine learning algorithms have proven to be of great practical value, especially in:

- Data mining problems concerning large databases that may contain valuable implicit regularities that can be discovered automatically (for an overview of this topic, the reader is referred to the special issue on Intelligent Information Retrieval of the IEEE Intelligent Systems and Their Applications, vol. 14, no. 4, pp. 30-70).
- Poorly understood domains where humans might not have well-established, generic knowledge needed to develop effective algorithms (e.g. in learning to play games or in learning to interpret human facial expressions in terms of emotions).
- Domains where the program must dynamically adapt to changing conditions (e.g. see the special issue on Self-adaptive Software of the IEEE Intelligent Systems and Their Applications, vol. 14, no. 3, pp. 26-63).

However, most of the machine-learning algorithms require a special training phase whenever information is extracted (knowledge generalization), which makes on-line adaptation (sustained learning) difficult. Virtually all techniques discussed in this section (except instance-based learning) are not well suited for on-line learning. Hence, learning in dynamic environments is cumbersome (if possible at all) for most machine-learning methods. Another common problem is that, in general, machine-learning techniques are data oriented: they model the relationships contained in the training data set. In turn, if the employed training data set is not a representative selection from the problem domain, the resulting model may differ from actual problem domain. This limitation of machine learning methods is aided and abetted by the fact that most of them do not allow the use of a priori knowledge. Finally, machine-learning algorithms have difficulties in handling noise. Though many of them have some special provisions to prevent noise fitting, these may have a side effect of ignoring seldom occurring but possibly important features of the problem domain.

2. Case-Based Reasoning

During the 70s and 80s, one of the most visible developments in AI research was the emergence of rule-based expert systems (RBES). These programs were applied to more and more problem domains requiring extensive knowledge for very specific and rather critical tasks including hardware troubleshooting, geological exploration and medical diagnosis. In general, the RBES should be based upon a deep, explicit, causal model of the problem domain knowledge that enables them to reason using first principles. But whether the knowledge is shallow or deep, an explicit model of the domain must still be elicited and implemented. Hence, despite their success in many sectors, developers of RBES have met several critical problems. These can be summarized as proposed by Schank [16]:

1. Difficult and time-consuming construction of the intended knowledge base due to complex and time-consuming expert knowledge elicitation. This is especially the case with problem domains covering a broad range of knowledge.
2. Incapability of dealing with problems that are not explicitly covered by the utilized rule base. In general, rule-based expert systems are useful if the built-in knowledge is well formalized, circumscribed, established and stable.
3. If no learning facility is built into a rule-based expert system, any addition to the existing program requires a programmer intervention.

Solutions to these problems have been sought through better elicitation techniques and tools, improved development paradigms, knowledge modeling languages and ontologies, and advanced techniques and tools for maintaining systems. However, in the past decade an alternative reasoning paradigm and computational problem-solving method attracted a great deal of attention: *Case-Based Reasoning* (CBR) solves new problems by adapting previously successful solutions to similar problems. CBR draws attention because it seems to address the problems outlined above directly [19]:

- CBR does not require an explicit domain model and so elicitation becomes a task of gathering case histories.
- Implementation is reduced to identifying significant features that describe a case, an easier task than creating an explicit model.
- CBR systems can learn by acquiring new knowledge as cases. This and the application of database techniques makes the maintenance of large volumes of information easier.

The work of Roger Schank [15], [16], is widely held to be the origin of CBR. He proposed a different view on model-based reasoning inspired by human reasoning and memory organization. Schank suggests that our knowledge about the world is mainly organized as memory packets holding together particular episodes from our lives that were significant enough to remember. These *memory organization packets* (MOPs) and their elements are not isolated but interconnected by our expectations as to the normal progress of events (called *scripts* by Schank). In turn, there is a hierarchy of MOPs in which “big” MOPs share “small” MOPs. If a MOP contains a situation where some problem was successfully solved and the person finds himself in a similar situation, the previous experience is recollected and the person can try to follow the same steps in order to reach a solution. Thus, rather than following a general set of rules, reapplying previously successful solution schemes in a new but similar context solves the newly encountered problems. Using these observations about human reasoning process Schank proposed *memory-based reasoning model* and memory-based expert systems [16] , which are characterized as follows:

- The utilized knowledge base is derived primarily from enumeration of specific cases or experiences. This is founded upon the observation that human experts are much more capable of recalling experiences than of articulating internal rules.
- As problems are presented to a memory-based expert system to which no specific case or rule can match exactly, the system can reason from more general similarities to come up with an answer. This is founded upon the generalization power of human reasoning. In general, we are reminded of something by the similarity, but the retrieval can be also based on differences. Furthermore, the retrieval is almost never full breadth and is highly context dependent. The reason for not performing an exhaustive recall is not only due to the cumbersomeness of such a task but also due to the organizations of MOPs: once we focus on some MOP it is very easy to recall other MOPs related to it by some features.
- The memory of experiences utilized by the system is changed and augmented by each additional case that is presented. A cornerstone of the memory-based model of reasoning is automatic learning: the system should remember the problems that it has encountered and use that information to solve future problems. This is founded upon the capability of the human brain to merge the progress of events seamlessly into the previously developed scripts of events.

The area of AI concerned with case-based reasoning puts Schank's memory-based reasoning model in practice. In a nutshell, CBR is reasoning by remembering: previously solved problems (cases) are used to suggest solutions for novel but similar problems. Kolodner [6] lists four assumptions about the world around us that represent the basis of the CBR approach:

1. *Regularity*: the same actions executed under the same conditions will tend to have the same or similar outcomes.
2. *Typicality*: experiences tend to repeat themselves.
3. *Consistency*: small changes in the situation require merely small changes in the interpretation and in the solution.
4. *Adaptability*: when things repeat, the differences tend to be small, and the small differences are easy to compensate for.

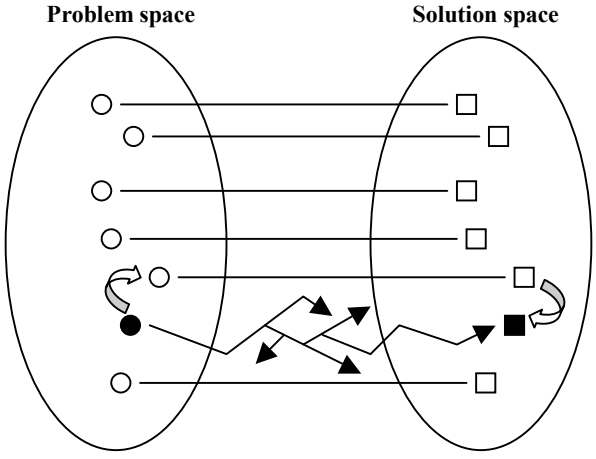

Fig.1: Problem solving using CBR

Fig. 1 illustrates how the assumptions listed above are used to solve problems in CBR. Once the currently encountered problem is described in terms of previously solved problems, the most similar solved problem can be found. The solution to this problem might be directly applicable to the current problem but, usually, some adaptation is required. The adaptation will be based upon the differences between the current problem and the problem that served to retrieve the solution. Once the solution to the new problem has been verified as correct, a link between it and the description of the problem will be created and this additional problem-solution pair (case) will be used to solve new problems in the future. Adding of new cases will improve results of a CBR system by filling the problem space more densely.

2.1 The CBR working cycle

In the problem solving illustrated in Fig. 1 and explained above, the following steps were taken: describing the current problem, searching for a similar previously solved problem, retrieving the solution to it, adapting the solution to the current problem, verifying the solution, and storing the newly solved problem. In turn, since the newly found solution may be used for solving future problems, the process illustrated in Fig. 1 denotes, in fact, the CBR working cycle.

According to Kolodner, the CBR working cycle can be described best in terms of four processing stages:

1. *Case retrieval*: after the problem situation has been assessed, the best matching case is searched in the case base and an approximate solution is retrieved.
2. *Case adaptation*: the retrieved solution is adapted to fit better the new problem.
3. *Solution evaluation*: the adapted solution can be evaluated either *before* the solution is applied to the problem or *after* the solution has been applied. In any case, if the accomplished result is not satisfactory, the retrieved solution must be adapted again or more cases should be retrieved.
4. *Case-base updating*: If the solution was verified as correct, the new case may be added to the case base.

Aamodt and Plaza [1] give a slightly different scheme of the CBR working cycle comprising the *four REs* (Fig. 2):

1. RETRIEVE the most similar case(s);
2. REUSE the case(s) to attempt to solve the current problem;
3. REVISE the proposed solution if necessary;
4. RETAIN the new solution as a part of a new case.

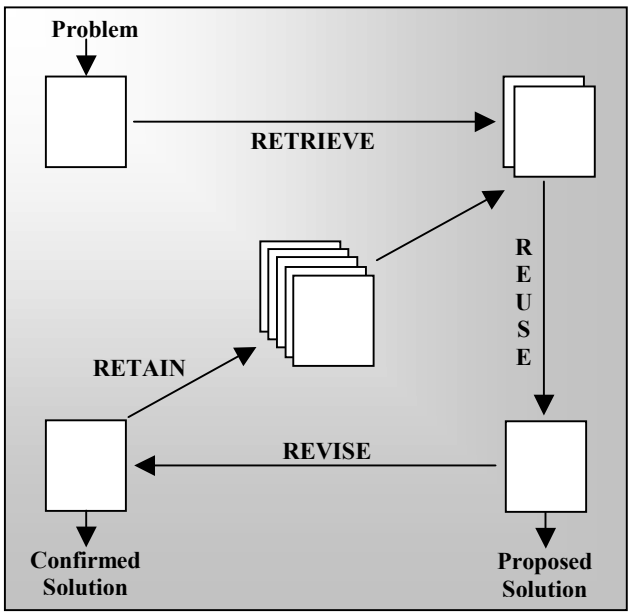

Fig. 2: The CBR cycle

Although they use different terminologies, the CBR working cycles denoted above are essentially the same. A new problem is matched against the cases furnishing the case base and one or more similar cases are *retrieved*. A solution suggested by the matching cases is then *reused*. Unless the retrieved case is a close match, the solution will probably have to be *revised* (*adapted*) and tested (*evaluated*) for success, producing a new case that can be *retained* ensuing, consequently, *update of the case base*.

2.2 Types of knowledge in CBR

CBR systems make use of many types of knowledge about the problem domain for which they are designed. Richter identifies four knowledge containers [14]: the vocabulary, similarity measures, adaptation knowledge, and cases themselves. The first three containers usually represent general knowledge about the problem domain. If there are any exceptions from this knowledge, they are commonly handled by appropriate cases.

Vocabulary includes the knowledge necessary for choosing the features utilised to describe the cases. Case features have to be specified so that they satisfy both: (i) being helpful in retrieving other cases, which contain useful solutions to similar problems, and (ii) being discriminative enough to prevent retrieval of too different cases, which could lead to false solutions and/or reduced performance. A thorough comprehension of the problem domain is necessary to be able to choose which of all problem parameters are the best as case features. In addition, either the vocabulary should be chosen such that it anticipates future expansion of the case base, or the system should be developed such that it alleviates automatic expansion of the vocabulary. Otherwise, it may be impossible to represent new problem features, which will then either be mapped to the available descriptors or be ignored, probably leading in both cases to wrong solutions.

Similarity measures include the knowledge about the similarity measure itself and the knowledge used to choose the most efficient organisation of the employed case base and the most appropriate case-retrieval method. For any given problem, there are many possible similarity measures that can be used. Hence, choosing the best among the available possibilities and implementing the chosen similarity measure efficiently exacts sound knowledge of the problem domain. This is especially important for classification problems involving complex structured cases since the value of the similarity can be used as a basis for automatic classification. As far as the organisation of the employed case base and the retrieval algorithm are concerned, a balance has to be found between case-memory models that preserve the semantic richness of cases and methods that simplify the access and retrieval of relevant cases. In general, knowledge about cases can be used to choose the organisational structure of the case base such that the cases can be accurately and efficiently retrieved.

Adaptation knowledge includes the knowledge necessary for implementing the adaptation and evaluation stages of the CBR working cycle. Generally, the adaptation stage requires knowledge about how differences in problems affect the solutions. This knowledge is usually coded in explicit rules. Yet, since for many problem domains, this is the most difficult knowledge to acquire, the adaptation is frequently left to the user of the system. This is especially the case when mistakes made by the system are expensive effecting the reliability of the system and, in turn, the user's confidence in it. Usually, before applying a new solution for solving a problem, its correctness has to be evaluated. The knowledge required for the evaluation stage concerns estimating the significance of differences and similarities between

the situations. Thus, this type of knowledge can be viewed as an extension and refinement of the knowledge furnishing the similarity measures container.

Cases contain knowledge about solved problem instances and, in many CBR systems, this represents the knowledge that the system acquires during use. What the cases will contain is mainly determined by the chosen vocabulary. Sometimes the employed case base is initialized with carefully selected cases that provide a problem domain coverage that is as even as possible (e.g., this is the case with the Facial Expression Analyzer described in [11]). This is commonly the case when the necessary adaptation stage is to be kept simple, yielding manageable system maintenance. Anyhow, new cases will usually be added during use. Yet, it is often unwise to store all the solved problems as cases. Large case bases may have high memory/storage requirements, may impose long retrieval times and, in turn, may reduce the system's performance. Therefore, heuristics should be specified for determining the useful cases to be stored in the case base.

2.3 Case representation

A case is a contextualized piece of knowledge representing an experience. It contains the past lesson that is the content of the case and the context in which the lesson can be used. In general, a case comprises a:

- *Problem description*, which depicts the state of the world when the case occurred;
- *Problem solution* which states the derived solution to that problem; and/or
- *Outcome*, which describes the state of the world after the case occurred.

Cases that comprise problems and their solutions can be used to derive solutions to new problems, whereas cases comprising problems and outcomes can be used to evaluate new situations. If such cases contain solutions in addition, they can be used to evaluate the outcome of proposed solutions and prevent potential problems. The more information is stored, the more useful the case can be. Yet entering all available information makes the system more complex and, in turn, more difficult to use. Due to these reasons, most of the CBR systems are limited to storing only problem descriptions and solutions.

The problem description essentially contains as much data about the problem and its context as necessary for an efficient and accurate case retrieval. Principally, it is useful to store retrieval statistics like the number of times the case was retrieved and the average match value. These statistics may be valuable for handling the case base: for prioritising cases, for pruning the case base by removing seldom used cases, and generally for maintenance of the case base.

The problem solution can be either atomic or compound. Atomic solutions are typical for CBR systems used for diagnosis or for classification in general. Compound solutions can be found for instance in CBR systems utilised for planning or design. A compound solution may be composed of a sequence of actions, an arrangement of components, etc. For example, in the case of the Facial Expression Analyzer reported in [11], compound solutions consist of multiple, user-defined, facial-expression interpretation labels. The main use of a solution is to serve as a starting point for educing new solutions. Therefore, the way a solution is derived may be of equal importance as that of the solution itself.

Cases can be represented as simple feature vectors, or they can be represented using any AI representational formalism such as frames, objects, predicates, semantic nets, or rules. The choice of particular representational formalism is largely determined by the information to be stored within a case. Cases can be monolithic or compound. Individual parts of compound cases can be processed or used separately. For example, a problem can be solved by reusing partial solutions from several compound cases, like explained in [11]. Most representational formalisms are proprietary for the more complex cases. Nevertheless, there is a lack of consensus within the CBR community as to exactly what information should be stored within a case and, in turn, which representational formalism should be used. However, two pragmatic measures can be taken into account in deciding both the information to be stored in a case and the appropriate representational formalism: the intended functionality and the ease of acquisition of the information represented in the case.

Finally, cases are the basis of any CBR system: a system without cases would not be a case-based system. Yet, a system using only cases and no other explicit knowledge (not even in the similarity measures) is difficult to distinguish from a nearest-neighbour classifier or a database retrieval system. In other words, such a system does not exploit the full generalisation power of CBR, resulting usually in poor system performance due to inefficient retrieval based upon case-by-case search of the whole case base.

2.4 Indexing

Within the CBR community, an explicit formal specification (i.e. ontology) of what the terms "indices" and "indexing" actually mean in terms of a CBR system has not been established yet. Kolodner identifies indexing with an accessibility problem [6], that is, with the whole set of issues inherent in setting up the case base and its retrieval process so that the right cases are retrieved at the right time. Thus, case indexing involves assigning indices to cases to facilitate their retrieval. CBR researches proposed several guidelines on indexing [19]. Indexes should be:

- *predictive* of the case relevance;
- *recognisable* in the sense that it should be understandable why they are used;
- *abstract* enough to allow for widening the future use of the case base;
- *concrete* (discriminative) enough to facilitate efficient and accurate retrieval.

Both manual and automated methods are used nowadays to select indices. Choosing indices manually involves deciding the purpose of a case with respect to the aims of the user and deciding under which circumstances the case will be useful. Kolodner claims that people tend to be better at choosing the indices than automatic algorithms. Anyhow, there is an ever increasing number of automated indexing methods. For a review of these the reader is referred to [19]. For an example of an automatic indexing algorithm performing indexing cases by (case) features that tend to be predictive across the entire problem domain, the reader is referred to [11].

Indices do not have to be rigid; they may change during use of the system. In fact, changing the indexes is one way of learning. Changes may be made if, for instance, a wrong case was retrieved or an entirely novel problem situation is encountered. Changes may involve changing weights (importance/priority) of the features, changing or adding features, changing

or adding pointers to other cases in the case base, etc. Similarly to selecting/generating the indexes, changing the indexes can be done either manually or automatically.

2.5 Case base organization

Case storage is an important aspect in designing efficient CBR system, in that it should reflect the conceptual view of what is represented in the case and take into account the indexes that characterise the case. As already mentioned above, the case base should be organised into a manageable structure that supports efficient and accurate search and retrieval methods. Accurate retrieval guarantees that the best matching case will be retrieved, and efficient retrieval guarantees that cases will be retrieved fast enough for acceptable system response times. These two factors are inversely proportional: it is easy to guarantee accurate retrieval at the expense of efficiency (e.g. by matching all the cases) and easy to have fast retrieval if only a fraction of the employed case base is searched (possibly missing some examples). Hence, a good case-base organisation and a good retrieval algorithm are the ones which yield the best compromise between accuracy and efficiency of the retrieval algorithm.

In general, three main approaches to case-base organisation can be distinguished: flat organisation, clustered organisation, and hierarchical organisation. Also a combination of these methods within the same case base is possible.

Flat organisation is the simplest case-base organisation that yields a straightforward flat structure of the case base. Though advantageous due to its simplicity and facile case addition/deletion, a flat case-base organisation imposes, in general, case retrieval based upon a case-by-case search of the whole case base. Hence, for medium and large case bases, this leads to time-consuming retrieval, yielding an inefficient CBR system.

Clustered organisation, originating in the dynamic memory model initially proposed by Schank [15], is the type of case-base organisation in which cases are stored in clusters of similar cases. The grouping of cases may be based on their mutual similarity (like in the case of the dynamic memory of experiences used by Pantic [10], [11]) or on the similarity to some prototypical cases. The advantage of this organisation is that the selection of the clusters to be matched is rather easy, as it is based upon the indexes and/or prototypical cases characterising the clusters. A disadvantage is that it needs a more complex algorithm for case addition/ deletion than a flat organised case base.

Hierarchical organisation, originating in the category-exemplar memory model of Porter and Bareiss [12], is the case-base organisation that is generally obtained when cases that share the same features are grouped together. The case memory is a network structure of categories, semantic relations, cases, and index pointers. Each case is associated with a category, while the categories are inter-linked within a semantic network containing the features and intermediate states referred to by other terms. Different case features are assigned different importance in describing the membership of a case to a category. It is important to note that this importance assignment is static; if it changes, the case-base hierarchy has to be redefined. A new case is stored by searching for a matching case and by establishing the relevant feature indexes. If a case is found with only minor differences to the new case, the new case is usually not retained. In turn, a hierarchical case-base organisation facilitates fast and accurate case retrieval. However, its higher complexity implies a rather cumbersome case

addition/deletion, potentially involving expensive case-base reorganisation and an inapt case-base evaluation and maintenance.

2.6 Retrieval

Given a description of a problem, a retrieval algorithm should retrieve cases that are most similar to the problem or situation currently presented to the pertinent CBR system. The retrieval algorithm relies on the indices and the organisation of the case memory to direct the search to case(s) potentially useful for solving the currently encountered problem.

The issue of choosing the best matching cases can be referred to as *analogy drawing*, that is, comparing cases in order to determine the degree of similarity between them. Many retrieval algorithms have been proposed in the literature up to date: induction search (e.g., ID3, [13]), nearest neighbour search, serial search, hierarchical search, parallel search, etc. (for examples, see [8]).

The simplest form of retrieval is the 1^{st}-nearest-neighbour search of the case base, which performs similarity matching on all the cases in the case base and returns just one best match [8]. It is to be expected that this method implies long retrieval times, especially in the case of large case bases. Therefore, cases are usually *preselected* prior to similarity matching. Cases can be preselected using a simpler similarity measure; commonly, this is done using the indexing structure of the case base. A typical problem with preselection concerns handling a situation where no best match has been found in the preselected set of cases; since preselection is merely approximate, there is a possibility that amongst the non-selected cases a better match can be found.

Another way of speeding up the retrieval is to employ *ranking of cases*. The simplest ranking method concerns exploiting the retrieval statistics for cases in the case base. The frequently retrieved cases can be considered as prototypical cases and probably should be matched first. Another ranking method is applicable to the clustered case-base organisation. It concerns matching the current case to the clusters' prototypes and then searching the matching clusters in the order determined by the degree of similarity between the matching clusters' prototypes and the current case.

The retrieval may result in retrieving single or *multiple best match cases*. In general, the retrieval mechanism tends to be simpler and faster if: (i) a larger number of possibly similar cases are retrieved, (ii) all of them are used to find solutions, and then (iii) the best solution is chosen. In this case, the retrieval algorithm itself may be less selective (and, therefore, simpler and faster) since the usefulness of the retrieved cases is to be determined in succeeding processing phases.

Finally, a way of speeding up the retrieval is to do it in parallel. A *parallel search* of the case base is realisable since case matching does not require exchange of much information between the parallel running processes. Thus, the speed gain scales up with the number of processing units. While the implementation of parallel retrieval is simple for flat and clustered case bases, it is rather difficult for hierarchical case bases. Though bringing significant speed gains, parallel retrieval is usually accompanied by an increase in implementation costs and software complexity.

2.7 Adaptation

Generally, once a matching case is retrieved, it will not correspond to exactly the same problem as the problem for which the solution is currently being sought. Consequently, the solution belonging to the retrieved case may not be optimal for the problem presently encountered and, therefore, it should be adapted. Adaptation looks for prominent differences between the retrieved case and the current case, and then (most commonly) applies a formulae or a set of rules to account for those differences when suggesting a solution. In general, there are two kinds of adaptation in CBR [19]:

1. *Structural adaptation* applies adaptation rules directly to the solution stored in cases. If the solution comprises a single value or a collection of independent values, structural adaptation can include modifying certain parameters in the appropriate direction, interpolating between several retrieved cases, voting, etc. However, if there are interdependencies between the components of the solution, structural adaptation requires a thorough comprehension and a well-defined model of the problem domain.
2. *Derivational adaptation* reuses algorithms, methods, or rules that generated the original solution to produce a new solution to the problem currently presented to the system. Hence, derivational adaptation requires the planning sequence that begot a solution to be stored in memory along with that solution. This kind of adaptation, sometimes referred to as *reinstantiation*, can only be used for problem domains that are well understood.

An ideal set of rules must be able to generate complete solutions from scratch, and an effective and efficient CBR system may need both structural adaptation rules to adapt poorly understood solutions and derivational mechanisms to adapt solutions of cases that are well understood. However, one should be aware that complex adaptation procedures make the system more complex but not necessarily more powerful. Complex adaptation procedures make it more difficult to build and maintain CBR systems and may also reduce system reliability and, in turn, user's confidence in the system if faulty adaptations are encountered due to, for example, incompleteness of the adaptation knowledge, which is the most difficult kind of knowledge to acquire [7]. Therefore, in many CBR systems, adaptation is done by the user rather than by the system. Mark et al. report that in a well-designed system, the users do not perceive "manual" adaptation as something negative [7].

2.8 Vantages and limitations of CBR

CBR is a lazy problem-solving method and shares many characteristics with other lazy problem-solving methods, including advantages and disadvantages. Aha [2] defines the peculiarities of lazy problem-solving methods in terms of three Ds:

- *Defer*: lazy problem solvers simply store the presented data and generalizing beyond these data is postponed until an explicit request is made.
- *Data-driven*: lazy problem solvers respond to a given request by combining information from the stored data.
- *Discard*: lazy problem solvers dismiss any temporary (intermediate) result obtained during the problem solving process.

Unlike lazy problem solvers, eager problem-solving methods try to extract as much information as possible from the presented data and then to discard the data prior to the actual

problem solving. An example of a lazy problem solver is a CBR classifier, while an ANN classifier is an example of an eager problem solver. Eager algorithms can be referred to as knowledge compilers, as opposed to lazy algorithms, which perform run-time knowledge interpretation. This is the key difference between lazy and eager problem solvers, which can also be explained by the following:

- Lazy methods can consider the current query instance $\boldsymbol{x}$ when deciding how to generalise beyond the training data (which have already been presented).
- Eager methods cannot, because their global approximation to the target function has already been chosen by the time they observe the current query instance $\boldsymbol{x}$.

To summarise, lazy methods have the option of selecting a different hypothesis or local approximation to the target function for each presented query instance. Eager methods using the same hypothesis space are more restricted because they must choose their approximation before the presented queries are observed. Consequently, a lazy method will generally require less computation during training, but more computation when it must generalise from training data by choosing a hypothesis based on the training examples near the currently presented query.

The benefits of CBR as a lazy problem-solving method are:

- *Ease of knowledge elicitation*: Lazy methods, in general, can utilise easily available cases or problem instances instead of rules that are difficult to extract. So, classical knowledge engineering is replaced by case acquisition and structuring [2].
- *Absence of problem-solving bias*: Because cases are stored in a "raw" form, they can be used for multiple problem-solving purposes. This in contrast to eager methods, which can be used merely for the purpose for which the knowledge has already been compiled.
- *Incremental learning*: A CBR system can be put into operation with a minimal set of solved cases furnishing the case base. The case base will be filled with new cases as the system is used, increasing the system's problem-solving ability. Besides simple augmentation of the case base, new indexes and clusters/categories can be created and the existing ones can be changed. This in contrast to virtually all machine-learning methods (see part 1 of this syllabus), which require a special training period whenever information extraction (knowledge generalisation) is performed. Hence, dynamic on-line adaptation to a non-rigid environment is possible [8].
- *Suitability for complex and not-fully formalised solution spaces*: CBR systems can be applied to an incomplete model of problem domain; implementation involves both to identify relevant case features and to furnish, possibly a partial case base, with proper cases. In general, because they can handle them more easily, lazy approaches are often more appropriate for complex solution spaces than eager approaches, which replace the presented data with abstractions obtained by generalisation.
- *Suitability for sequential problem solving*: Sequential tasks, like these encountered in reinforcement learning problems, benefit from the storage of history in the form of a sequence of states or procedures. Such a storage is facilitated by lazy approaches.
- *Ease of explanation*: The results of a CBR system can be justified based upon the similarity of the current problem to the retrieved case(s). Because solutions generated by CBR are easily traceable to precedent cases, it is also easier to analyse failures of the system. As noted by Watson and Marir [19], the explanations provided based upon individual and generalised cases tend to be more satisfactory than explanations generated by chains of rules.
- *Ease of maintenance*: This is particularly due to the fact that CBR systems can adapt to many changes in the problem domain and the pertinent environment, merely by acquiring

new cases. This eliminates some need for maintenance; only the case base(s) needs to be maintained.

Major disadvantages of lazy problem solvers are their memory requirements and time-consuming execution due the processing necessary to answer the queries. The limitations of CBR can be summarised as follows:

- *Handling large case bases*: High memory/storage requirements and time-consuming retrieval accompany CBR systems utilising large case bases. Although the order of both is at most linear with the number of cases, these problems usually lead to increased construction costs and reduced system performance. Yet, these problems are less and less significant as the hardware components become faster and cheaper.
- *Dynamic problem domains*: CBR systems may have difficulties in handling dynamic problem domains, where they may be unable to follow a shift in the way problems are solved, since they are usually strongly biased towards what has already worked. This may result in an outdated case base.
- *Handling noisy data*: Parts of the problem situation may be irrelevant to the problem itself. Unsuccessful assessment of such noise present in a problem situation currently imposed on a CBR system may result in the same problem being unnecessarily stored numerous times in the case base because of the difference due to the noise. In turn this implies inefficient storage and retrieval of cases.
- *Fully automatic operation*: In a typical CBR system, the problem domain is usually not fully covered. Hence, some problem situations can occur for which the system has no solution. In such situations, CBR systems commonly expect input from the user.

2.9 CBR application domains

Although CBR is a relatively new AI methodology, numerous successful applications exist in the academic as well as in the commercial domain. Already in 1994, Watson and Marir reported over 100 commercially available CBR applications [19]. The domains of these numerous CBR systems reported in the literature are the following:

- *Interpretation* as a process of evaluating situations/problems in some context (e.g., HYPO for interpretation of patent laws proposed in 1991, KICS for interpretation of building regulations proposed in 1994, LISSA for interpretation of non-destructive test measurements proposed in 1999).
- *Classification* as a process of explaining a number of encountered symptoms (e.g., CASEY for classification of auditory impairments proposed in 1989, CASCADE for classification of software failures proposed in 1992, PAKAR for causal classification of building defects proposed in 1994, ISFER for classification of facial expressions into user-defined interpretation categories proposed in [10]).
- *Design* as a process of satisfying a number of posed constraints (e.g., JULIA for meal planning proposed in 1992, Déjà Vu for control-software production proposed in 1996, CLAVIER for design of optimal layouts of composite airplane parts proposed in 1996, EADOCS for aircraft panels design proposed 1997).
- *Planning* as a process of arranging a sequence of actions in time (e.g., BOLERO for building diagnostic plans for medical patients proposed in 1993, TOTLEC for manufacturing planning proposed in 1993).
- *Advising* as a process of resolving diagnosed problems (e.g., DECIDER for advising students proposed in 1987, HOMER – a CAD/CAM help desk proposed in 1998).

References

[1] A. Aamodt and E. Plaza, "CBR: foundational issues, methodological variations and system approaches", *AI Communications*, vol. 7, no. 1, pp. 39-59, 1994.

[2] D.W. Aha, "The omnipresence of case-based reasoning in science and application", *Knowledge-Based Systems*, vol. 11, no. 5-6, pp. 261-273, 1998.

[3] C.G. Atkeson, A.W. Moore and S. Schaal, "Locally Weighted Learning", *Artificial Intelligence Review*, vol. 11, pp. 11-73, 1997.

[4] P. Clark and R. Niblett, "The CN2 induction algorithm", *Machine Learning*, vol. 3, pp. 261-284, 1989.

[5] T. Dietterich, H. Hild, and G. Bakiri, "A comparison of ID3 and backpropagation for English text-to-speech mapping", *Machine Learning*, vol. 18, no. 1, pp. 51-80, 1995.

[6] J. Kolodner, "Making the implicit explicit: Clarifying the principles of case-based reasoning", *Case-Based Reasoning: Experiences, Lessons & Future Directions*, D.B. Leake, (Ed.), pp. 349-370, AAAI Press, Menlo Park, USA, 1996.

[7] W. Mark, E. Simoudis and D. Hinkle, "Case-based reasoning: Expectations and results", *Case-Based Reasoning: Experiences, Lessons & Future Directions*, D.B. Leake, (Ed.), pp. 269-294, AAAI Press, Menlo Park, USA, 1996.

[8] T.M.. Mitchell, *Machine Learning*. Singapore: McGraw-Hill Companies Inc., 1997.

[9] M. Negnevitsky, *Artificial Intelligence: A Guide to Intelligent Systems*. Essex, UK: Addison Wesley, 2nd edition, 2004.

[10] M. Pantic, *Facial Expression Analysis by Computational Intelligence Techniques*. PhD thesis, Delft University of Technology, 2001.

[11] M. Pantic and L.J.M. Rothkrantz, "Case-based reasoning for user-profiled recognition of emotions from face images", *Proc. IEEE Int'l Conf. Multimedia and Expo*, 2004.

[12] B.W. Porter and E.R. Bareiss, "PROTOS: Experiment in knowledge acquisition for heuristic classification tasks", *Proc. 1st Int'l Meeting on Advances in Learning*, pp. 159-174, 1986.

[13] J.R. Quinlan, *Programs for Machine Learning*. San Mateo, USA: Morgan Kaufmann, 1993.

[14] M.M. Richter, "On the notion of similarity in case-based reasoning", *Mathematical and Statistical Methods in Artificial Intelligence*, G. della Riccia, R. Kruse, R. Viertl, (Eds.), pp. 171-184. Heidelberg, Germany: Springer-Verlag, 1995.

[15] R.C. Schank, *Dynamic memory: A theory of reminding and learning in computers and people*. Cambridge, UK: Cambridge University Press, 1982.

[16] R.C. Schank, *Memory-based expert systems*. Technical Report (# AFOSR. TR. 84-0814), Yale University, New Haven, USA, 1984.

[17] N. Sebe, M.S. Lew, I. Cohen, Y. Sun, T. Gevers and T.S. Huang, "Authentic Facial Expression Analysis", *Proc. IEEE Int'l Conf. Face and Gesture Recognition*, pp. 517-522, 2004.

[18] R.S. Sutton and A.G. Barto, *Reinforcement learning: An introduction*. Cambridge, USA: MIT Press, 1998.

[19] I. Watson and F. Marir, "Case-base reasoning: A review", *The Knowledge Engineering Review*, vol. 9, no. 4, pp. 327-354, 1994.