Data Mining: The Search for Knowledge in Databases

M. Holsheimer and A. Siebes, Notes taken by *Jiawei Han*

1 Introduction

1. Deduction: infers information that is a *logic consequence* of the information in the database. E.g., join can be viewed as an operator for deduction. More is researched in deductive databases.

2. Induction: infers information that is *generalized* from the information in the database. A *high level* summary (*knowledge, regularities*) of information in the database.

3. Inductive learning: groups similar objects in classes and constructs rules that predict the behavior of the objects of such a class.

   Supervised learning: An external teacher defines classes and provides the system with examples of each class. The system discovers common properties in the examples for each class – the class description (*learning from examples*). A class, together with its class description forms a classification rule '*if* ⟨*description*⟩ *then* ⟨*class*⟩' that can be used to predict the class of previously unseen objects.

   Unsupervised learning: The system has to discover the class itself, based on common properties of objects (*learning from observation and discovery*).

4. Machine learning: Training set $\rightarrow$ iterative search strategy (hypothesis, quality function, accepted-rejected-improved). When multiple alternatives exist for the new hypothesis, the choice is guided by *heuristics, constraints*, and *user-interaction*.

5. Data mining:

   ⋄ Data set: large, noisy, missing data; represents only a small set of all possible behavior; and cannot manipulate its environment.

   ⋄ Browsing optimization, caching, statistical techniques.

   ⋄ Difference from statistical techniques: assists users in the generation of hypotheses.

2 Inductive Learning

2.1 Models

1. Environment: The environment is a state transition system, i.e., a pair of $(\mathcal{S}, \mathcal{T})$, where $\mathcal{S}$ is the set of all possible states and $\mathcal{T}$ is the transition function $\mathcal{T} : \mathcal{S} \rightarrow \mathcal{S}$.

   Assume objects in an environment are completely unrelated (???): an absurd assumption (???) based on the belief: most data mining systems search for relationships within objects (i.e., properties in a single object) (Is it ture???).

2. Classes: The set of all classes is denoted $\mathcal{C}$, and to each class C_i corresponds a description D_i of values for the selected properties. A classification function P: $S \rightarrow \mathcal{C}$, that maps an object o in state S to class C_i when o satisfies D_i.

3. Model transition function: With this internal representation, a model transition function $\mathcal{T}'$ is intended to mimic the transition function $\mathcal{T}$, operating in the real world.

4. Correctness of the model: A model is correct if it predicts the next state correctly, i.e., $P(\mathcal{T}(S_t)) = \mathcal{T}'(P(S_t))$.

2.2 Learning

1. Learn: finding both an appropriate internal representation, i.e., classes, and a model transition function, acting on this representation.

   ⋄ *Learning by being told*: translates the acquired (from teacher, textbook, etc.) knowledge to an internal format.

   ⋄ *Learning from analogy*: transforms existing rules to make them applicable to new, but similar situations.

   ⋄ *Inductive learning*: is more useful in data mining.

2.2.1 Supervised Learning: (Learning from examples)

1. Teacher defines classes and supplies examples (i.e., pre-classified objects) of each class. The system finds the descriptions for each class.

2. Single class learning: Characteristic description (single out instances of class C from those not an instance of C) – *positive* examples, *negative* examples (especially, *near-misses*).

3. Multiple class learning: The teacher defines a finite number of classes $C_1, C_2, \ldots, C_m$, for which (*characteristic* or *discriminating*) descriptions have to be found.

2.2.2 Unsupervised Learning: (Learning from observation and discovery)

1. The system has to find its own classes (by clustering), class descriptions (one for each discovered class, and together covering all of the objects in the environment) – form a high level summary of the objects in the environment.

2.3 Quality

1. We must distinguish between really existing relationships among states and apparent relationships that are not generally valid, but occur only because of the limited number of examples.

2. If we can construct multiple models, choose the simplest, because it is more likely to capture the nature of the phenomenon (Ockham's razor).

3 Data mining

3.1 Induction from databases

3.1.1 Machine learning vs. data mining

1. Database is often designed for purposes different from data mining. Thus properties or attributes that would simplify the learning task are not necessarily present and cannot be requested from the real world.

2. Data set: noisy (errors), missing or contradictory data, represents only a small set of all possible behavior, and cannot manipulate its environment.

3.1.2 Training set

1. Assumption 1: Tuples in DB represent only properties of the objects but not relationships between those objects (?? is this true?).

2. Assumption 2: Universal relation assumption.

3. Let $\mathcal{A} = \{A_1, \cdots, A_n\}$ be a set of attributes with domains $Dom_1, \cdots, Dom_n$. A training set is a table over $\mathcal{A}$. An example is a tuple in a training set. The universer $\mathcal{U}$ is the full relation over $\mathcal{A}$. Hence, each training set is a finite subset of $\mathcal{U}$.

3.1.3 Classes

1. A DB contains one or more attributes that denote the class (concept) of a tuple, called *predicted attributes*. The remaining ones called *predicting attributes*.

2. A class C_i is a subset of the training set S, consisting of all objects that satisfy the class condition $cond_i$.

$$C_i = \{o \in S | cond_i(o)\}$$

   Objects that satisfy $cond_i$ are *positive* examples or instances of the class C_i. The examples outside this subset of the training set are *negative* examples.

3. Learning classification rules: Find the rules that predict the class from the predicting attributes. User defines conditions for each class C_i by partitioning S into subsets C_i ($i = 1, \ldots, n$). The system constructs descriptions D_i for these classes.

4. Conditions on attributes: Define a class in terms of the predicted and possibly some predicting attributes, e.g., "diagnosis = flu", "income > expenses".

5. Multiple databases, database over time, etc.

3.1.4 Clustering

1. Unsupervising learning: The system should first discover subsets of related objects in the training set and then find descriptions describing these subsets.

2. The number of possible clusterings of a database with N tuples: ($P(N,m)$ is the number of ways to partition N elements in m non-empty disjoint subsets).

$$C(N) = \Sigma_{m=1}^{N} P(N,m) = \Sigma_{m=1}^{N} \frac{1}{m!} \Sigma_{j=0}^{m} \binom{m}{j} (-1)^j (m-j)^N.$$

   Even if we assume that the number of classes in the database is small, say 10, the number of possible clusterings, i.e., $\Sigma_{m=1}^{10} P(N,m)$ is still staggering.

3.1.5 Rules

1. Description: Let $\mathcal{A}$ be the set of predicting attributes. An *elementary description* is a formula $A_1 = c_1 \wedge \cdots \wedge A_n = c_n$ such that:
   1. $A_i \in A$ *and* $i \neq j \rightarrow A_i \neq A_j$;
   2. $c_i \in Dom(A_i)$.

   A pattern or *description* is a (non-empty) disjunction of elementary descriptions. A condition $A_i = c_i$ is called an *attribute-value condition*. The set of all possible descriptions is called the *description space* and is denoted by $\mathcal{D}$.

2. Number of elementary descriptions $= \Pi_{\mathcal{A}} (|Dom(A_i)| + 1)$.

   The fraction of (non-empty) subset of $\mathcal{U}$ that can be described with elementary descriptions is
$$\frac{\Pi_{\mathcal{A}} (|Dom(A_i)| + 1)}{2^{\Pi_{\mathcal{A}} |Dom(A_i)|} - 1} \approx \frac{|\mathcal{U}|}{2^{|\mathcal{U}|}}.$$

3. Rule: A classification rule consists of a *description* D and a *class symbol* C:
$$\forall o \in \mathcal{U} : o \in \sigma_D(S) \rightarrow o \in C.$$
   i.e., any object that satisfies D belongs to class C.

3.2 Computational Learning Theory

4 Search Algorithms

4.1 Search Space

1. The search space $\langle \mathcal{D}, f, \mathcal{O} \rangle$ consists of a set of descriptions $\mathcal{D}$, a set of operations on these descriptions $\mathcal{O}$ and a quality function f.

2. Description space $\mathcal{D}$ is the set of all constructible descriptions, in a particular presentation. Each description D in $\mathcal{D}$ corresponds a subset of the training set S, caller the *cover* $\sigma_D(S)$.

3. Operations:

   (a) Generalization: 'Weaken' the description D, i.e., make it cover more objects: $\sigma_D(S) \subseteq \sigma_{D'}(S)$. Not truth preserving but falsity preserving.

   Generalization: A set-description $D = (A_1 \in S_1 \wedge \cdots A_i \in S_i \wedge \cdots A_n \in S_n)$ is generalized by extending the set of values for a particular attribute A_i to S_i' where $S_i \subset S_i' \subseteq Dom$.

   Special cases: dropping conditions, climbing generalization tree (where Dom_i forms a hierarchy and a subset is subsumed by a symbol).

   (b) Specialization: 'Strengthen' the description, i.e., make it cover less objects: $\sigma_D(S) \supseteq \sigma_{D'}(S)$.

4. Domain of the attributes: The structure of the domain can be one of the following basic types.

   (a) Nominal (categorical): independent symbols or names, unordered.

   (b) Linear: totally ordered, e.g., a numerical domain. This allows us to use intervals instead of sets of values. Linear domains are either (1) *ordinal*, such as {low, medium, high}, where an ordering is defined but other operations make no sense, (2) *interval*, where addition is defined but not multiplication, e.g., 20C is not twice warm as 10C; or (3) *ratio*, (such as numbers) where both addition and multiplication have useful interpretations.

   (c) Partially ordered: forming a hierarchy, where a parent node denotes a more general concept than its children.

5. Quality function: assigns a value, e.g., in the domain [0,1], to each description, indicating its quality.

   (a) Validity: relies on Ockham's razor: the simpler a description, the more likely it is that it describes some really existing relationship in the database. This complexity can be measured, e.g., in the size of the description. The validity f_v is higher for simpler descriptions.

   (b) Correctness, supervised learning.

      i. A description D for a class C is correct if it covers all positive, and none of the negative examples, i.e., if $\sigma_D(S) = C$.

ii. Classification accuracy: the probability that an object covered by the description actually belongs to the class, i.e., the relative portion of $\sigma_D(S)$ that is also covered by C.

$$classification\ accuracy = \frac{|\sigma_D(S) \cap C|}{|\sigma_D(S)|}$$

iii. Coverage: the probability that an arbitrary object, belonging to the class C, is covered by the description D.

$$coverage = \frac{|\sigma_D(S) \cap C|}{C}$$

iv. **Complete rules:** If the coverage is 1, the rule is complete. $C \subseteq \sigma_D(S)$. (D is a necessary condition of the class).

Deterministic rules: If the classification accuracy is 1, the rule is deterministic. $C \supseteq \sigma_D(S)$. (D is a sufficient condition of the class).

Correct rules: If both the classification accuracy and coverage is 1, the rule is correct. D is both a necessary and sufficient condition of the class.

v. Correctness of a rule (by Piatetsky-Shapiro'91).

- $f_c = 0$ if D and C are statistically independent. If the probability that a particular example covered by description D belongs to class C as well is equal to the probability that an arbitrary example in S belongs to C, i.e.,

$$\frac{|\sigma_D(S) \cap C|}{|\sigma_D(S)|} = \frac{|C|}{|S|}$$

then D and C are statistically independent, and the description is not interesting.

- f_c monotonically increases with $|\sigma_D(S) \cap C|$: the number of tuples in the overlapped region, when $\sigma_D(S)$ and C remain the same.
- f_c monotonically decreases with $|\sigma_D(S)|$ or $|C|$ when $\sigma_D(S) \cap C$ remains the same.
- The simplest function satisfying the principles is:

$$|\sigma_D(S) \cap C| - \frac{|\sigma_D(S)||C|}{|S|}$$

i.e., the difference between the actual number of examples of which the rule classifies correct and the number expected if C were independent of D.

(c) Correctness, unsupervised learning.

i. When the training set contains only positive examples, or when no classes are defined by the user (i.e., unsupervised learning), application of the above correctness criterion will result in over-generalization. Any simple description (e.g., *true* that covers the entire training set) will be assigned a high quality.

ii. Michalski and Step suggest that a quality function should not only depend on the simplicity of the description, but also on the *fitness*: how close does the description approximate the set of examples, that is, $\frac{|\sigma_D(S)|}{|\sigma_D(\mathcal{U})|}$, where $|\sigma_D(\mathcal{U})|$ is the total number of examples in the universe, covered by the description D. This ratio is exactly one if the description D covers only examples in the training set S, but none of the examples outside S.

(d) Combining criteria:

i. The quality of a description depends on its validity f_v, correctness f_c, and some other factors, e.g., the cost of evaluating the description, or the cost of measuring attributes used in the description, etc.

ii. We can combine these criteria to compute the overall quality: (1) assign a weight to each criterion and get a weighted sum, or (2) use a lexicographic evaluation function (LEF): LEF = $\langle (f_1, t_1), (f_2, t_2), \ldots (f_n, t_n) \rangle$, where t_1, t_2, ..., t_n are tolerances.

4.2 Search algorithm

4.2.1 Initial description

1. Bottom-up (data-driven): [DBLearn] started with the set of all examples in the target class, modified by repeated generalizations, which results in a more general rule that classifies the examples correctly (or correct within a certain tolerance).

2. Top-down (model-driven): [Meta-Dendral, Bacon, ID-3] Choose an initial description (could be the entire Universe), transform it by applying a sequence of generalization and specialization operations until its quality exceeds the threshold.

4.2.2 Search graph

1. To the initial description D_1, some of the operations in $\mathcal{O}$, say $o_1, o_2, \ldots, o_n$ can be applied, resulting in new descriptions D_{2i}, which may result in newer descriptions D_{3j}, ... Hence a search graph is formed.

2. Search strategies: Irrevocable vs. tentative (backtracking), depth-first vs. breadth-first.

4.3 Heuristic Search

Reduce search effort by carefully selecting operations such that a description of sufficient quality is found as soon as possible.

4.3.1 Hill climbing

1. Choose always the operations that result in the greatest quality improvement.

2. Expectation function (returns the quality of the description) $\mathcal{F} : \mathcal{D} \times \mathcal{O} \rightarrow [0, 1]$ is defined as

$$F(D, o) = f(o(D))$$

   where f is the quality function, and $o(D)$ is the description, resulting from application of operation o on D.

3. Generalize the expectation function by looking n levels ahead (i.e., the maximum quality of the descriptions that can be constructed within n steps):

$$F^n(D, o) = max_{o_i \in \mathcal{O}} F^{(n-1)}(o(D), o_i)$$

4. Instead of computing the quality of all possible extensions, one can also *estimate* the probability that applying a particular operation will result in a description of sufficient quality, e.g., ID3.

4.3.2 Limitations on the operations

Domain knowledge (specific to a particular domain, supplied by the user) can be used to guide the search process. Michalski distinguishes the following domain knowledge.

1. Irrelevant attributes: Remove irrelevant attributes.

2. Interrelationship between attributes: There may exist known relationships between attributes that constrain their set of possible values.

4.3.3 User interaction

User selects one or more descriptions for further investigation.

4.3.4 Previously discovered rules and classes

Previously discovered rules and classes can be used to guide the search process (incremental learning).

4.4 Alternative search algorithms

4.4.1 Genetic algorithms

1. Originated from the studies of cellular automata (Holland et. al.). A search procedure modeled on the mechanics of natural selection rather than a simulated reasoning process.

2. Basic idea: to use a set of candidate descriptions (called *population*), and to gradually improve the quality of this population by constructing new descriptions, assembled from parts of the best descriptions in the current population.

3. The candidate descriptions (organisms) are strings of symbols from a particular alphabet, e.g. $\{0, 1\}$. Each set-description is coded as a fixed length bit string. Two operations: crossover, mutation.

4. The Michigan approach (Holland): An organism represents a set-description.

5. The Pittsburgh approach: An organism is not a single description but a disjunction of set descriptions. The quality of organism can be computed using the quality function (based on the correctness, classification accuracy and coverage).

6. Genetic algorithms outperform traditional learning techniques, especially when the descriptions to be learned are complex or when no domain knowledge is available. However, GAs suffer two important drawbacks: (1) outperform traditional techniques only when almost no domain knowledge is available, (2) number of evaluations required is large.

4.4.2 Simulated annealing

1. Hill-climbing search has a local-maximum problem.

2. Simulated annealing: traverses the search space with a certain amount of arbitraries.

3. The probability of an operation being chosen depends on:

   ◇ Quality increase/decrease: Operations that increase the quality are assigned a higher probability.

   ◇ Temperature: If the temperature is high, all operations are chosen with more or less equal probability; if low, operations that increase the quality are preferred to those that decrease the quality; if zero, reduced to hill-climbing.

   ◇ Start with a high temperature (escape from local maxima), slowly decreases temperature and the search stabilizes in the global maximum.

5 Problems

5.1 Limited information

1. Incomplete information: If not all predicting variables are known, we may (1) restrict ourselves to deterministic rules (strong rules) – construct rules when all variables relevant to the classification are known; or (2) find probabilistic rules.

2. Sparse data: A database generally represents only a small subset of all possible behavior. Thus class boundaries cannot be determined exactly, and will be either vague or incorrect. Previously unseen objects are likely to be incorrectly classified. A solution could be *interacting with its environment* if possible, or *browsing for additional information.*

3. Samples: Assume that an arbitrary object in S belongs to class C with probability p, and that an object in set $\sigma_D(S)$ belongs to this class with probability p', and $p \neq p'$. Is the difference between p and p' statistically significant? — need statistic techniques to check the validity of discovered relationships.

4. Test set: The correctness can be tested by splitting the database into two sets: a *training set* (to construct descriptions) and a *test set* (to test the accuracy of these descriptions).

   For adaptive systems (systems that adjust their rules over time), use a sliding window, i.e., compute the actual probability of a rule over the last n classifications.

5.2 Data corruption

1. Noise:

   (a) Constructing descriptions. Quinlan uses statistical techniques to decide if an attribute should be added to the classification tree: a small amount of exceptional data is considered to be caused by noise, and can therefore be neglected.

   (b) Classifying examples. In some systems, adding noise to data resulted in *graceful degradation*: adding even substantial noise resulted in low levels of misclassification of unseen examples. Quinlan concluded that "it is not worthwhile expending effort to eliminate noise from the attribute values of objects if there is going to be a significant amount of noise when the induced classification rule is used in practice".

2. Missing attribute values:

   (a) Constructing descriptions. Examples with missing attribute values can be *discarded*, *replaced by "most-likely" value*, (predict the missing value based on values of other attributes in the example, and the class information), or *add the value "unknown"* to the domain of each attribute.

   (b) Classifying examples. The constructed rules can be used to classify unseen examples from which attribute values are missing. When a rule contains conditions on some of these attributes, one can compute the *probability* that a certain rule applies.

5.3 Databases

1. Size:

   (a) Information per object. E.g., in a medical information system, each visit is a record with 400 attributes, and 50 to 100 visits are stored for each patient.

   The number of descriptions depends on the size of the domains, e.g., using set descriptions, the number of constructible descriptions is roughly 2^l, where l is the sum of the domain-sizes of the attributes ($l \gg 1000$).

   (b) Number of objects: Large (typically $> 100,000$ objects).

   To overcome the cost problem, we can use: (1) *multiple* descriptions constructed in one pass, and (2) windows: compute the quality of a description using a representative sample of the database, called a *window*, and moreover, incremental updating the description can be performed.

2. Updates: A learning system should adapt to the changes to databases. Either reconstruct rules from scratch or perform *incremental learning.* Learning with full memory (remembering both old and new) vs. learning with partial memory. Value more recent information by assigning higher weights to recent experiences in the learning process.

6 Knowledge Representation

6.1 Propositional-like representation

1. Attribute-value conditions: e.g. '(color = red $\vee$ color = green) $\wedge$ shape = circle)' is a formula in CNF. The set-description 'color $\in$ {red, green} $\wedge$ shape $\in$ {circle}' is essentially a CNF.

   CNF involves a variable, the object itself, e.g., the above is actually, '$P(X) = ((X$.color = red $\vee$ X.color = green) $\wedge$ X.shape = circle)'.

2. Decision trees. ID3 (Quinlan). Nodes: attribute names, edges: possible values of the attribute, leaves: different classes. An object is classified by following a path down the tree. Disadvantages: decision trees tend to grow very large for realistic applications and are thus difficult to interpret by humans.

3. Production rules. Decision trees can be transformed into propositional-like production rules. Michalski's AQ15 uses production rules.

   Example. Each path in a decision tree corresponds to a term, e.g.,

   ```
   if outlook = sunny and humidity = high then class = N.
   if outlook = rain and windy = true then class = N.
   default class = P.
   ```

4. Decision lists: It strictly generalizes both decision trees, DNF and CNF representations. A decision list is a list of pairs
   $$(\phi_1, C_1), (\phi_2, C_2), \ldots, (\phi_r, C_r)$$
   where each ϕ_i is an elementary description, and each C_i is a class, and the last description ϕ_r is the constant *true*. The class of an object o is C_j if j is the least index of a description ϕ_j that covers object o. Such an index always exists, since the last term is always true, i.e., the default class. It can be viewed as an extended rule: "if ϕ_1 then C_1 elseif ϕ_2 ...else C_r".

5. Ripple-down rule sets. Using the nested if-then statements, giving all the exceptions locally, we can eliminate the need for a global ordering of the rules, as in decision lists. A ripple-down rule of depth 2 is shown below.

   if ϕ_i then
       if ϕ_j then C_j
           else C_i
       else C_j

6.2 First-order logic

1. Shortcomings with prepositional-like logic: restricted representation, inability to make use of domain knowledge, strong bias to vocabulary. Suggested direction: inductive logic programming. Quinlan's FOIL system, constraint logic programming, LIFE.

2. Expressive power vs. computational complexity. FOL more expressive, which may reduce the complexity. But more different descriptions can be constructed, and the description space extends. Solutions: finding particular solutions only, or developing good heuristics.

6.3 Structured representations: semantic nets and frames

6.4 Neural networks

1. A connectionist model: densely interconnected networks of simple computational elements. Multilayer perceptron, useful for implementing a classification function that assigning an object, denoted by an input vector $[x_0, x_1, \ldots, x_n]$, to one or more classes C_1, C_2, ..., C_m.

   Learning: compute the actual output vector and compare to the desired output. Then weights and thresholds are adjusted, using a back-propagation algorithm.

2. Neural nets vs. symbolic learning methods.

   ◇ Learning processes in neural nets are very slow, compared to symbolic learning systems (Quinlan: ID3 outperforms back propagation neural nets by a factor of 500 to 100,000).

   ◇ Knowledge, generated by neural nets, is not explicitly represented in the form of rules or conceptual patterns, but implicitly in the network itself, as a vast number of weights.

   ◇ It is difficult to incorporate any domain knowledge or user interaction in the learning process, not the case with data mining.

7 Overview of Data Mine Systems

7.1 ID3

1. Quinlan ('83, '86): Induction of Decision Trees. A supervised learning system that constructs decision trees from a set of examples (tuples).

2. Search space: Knowledge is represented as a decision tree. The quality (function) of a tree depends on both the classification accuracy and the size of the tree. Trees that classify all objects in the test set correctly, and are simple (Ockham's razor) as well, are preferred.

3. Search algorithm: The ID3 system uses a top-down irrevocable strategy that searches only part of the search space, guaranteeing that a simple – but not necessarily the simplest – tree is found.

   A tree (but not necessarily simple) is constructed as follows.

   (a) An attribute is selected as the root of the tree, and branches are made for all different values this attribute can have;

   (b) The tree is used to classify the training set. If all examples at a particular leaf belong to the same class, this leaf is labeled with this class. If all leaves are labeled with a class, the algorithm terminates.

   (c) Otherwise, the node is labeled with an attribute that does not occur on the path to the root, and branches are created for all possible values. Return to Step (3b).

   A simple tree can be generated by a suitable selection of attributes. An information-based heuristic selects the attribute providing the highest information gain, i.e., the attribute which minimizes the information needed in the resulting subtrees to classify the elements.

4. ID3 uses an information-theoretic approach aimed at minimizing the expected number of tests to classify an object. The attribute selection part of ID3 is based on the plausible assumption that the complexity of the decision tree is strongly related to the amount of information conveyed by this message.

   Let C contain p objects of class P and n objects of class N. An arbitrary object will be determined to belong to class P with probability $p/(p+n)$ and to class N with probability $n/(p+n)$. When a decision tree is used to classify an object, it returns a class. A decision tree can thus be regarded as a source of message 'P' or 'N', with the expected information needed to generate this message given by

$$I(p,n) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n}.$$

If attribute A with values $\{A_1, A_2, \cdots, A_m\}$ is used for the root of the decision tree, it will partition C into $\{C_1, C_2, \cdots, C_m\}$, where C_i contains those objects in C that have value A_i of A. Let C_i contain p_i objects of class P and n_i objects of class N. The expected information required for the tree with A as root is then obtained as the weighted average:

$$E(A) = \sum_{i=1}^{m} \frac{p_i + n_i}{p+n} I(p_i, n_i).$$

The information gained by branching on A is

$$gain(A) = I(p,n) - E(A).$$

ID3 examines all candidate attributes and choose A to maximize $gain(A)$, forms the tree, and then uses the same process recursively to form decision tree for the residual subset $C_1, C_2, \cdots, C_m$.

A known problem with this criterion is that it tends to favor attributes with many values. Different solutions to this problem are discussed in Quinlan('86).

5. Numerical attributes: test on ranges of numerical attributes (C4.5 – an extension to ID3) allows tests on the inequality of numerical attributes, such as $A_i \leq N$ and $A_i > N$, with two possible branches.

6. Grouping attribute values: test whether the value of an attribute belongs to a particular set of values, such as $A_i \in \{v_1, v_2, \ldots, v_n\}$. However, for m different values, there are $2^{m-1} - 1$ different binary partitions, which excludes the possibility of an exhaustive search for the best partitioning. C4.5 uses an irrevocable bottom-up search, based on iterative merging of groups.

7. Noise: To reduce the effect of corrupted examples, we can require branching on A if the information gain exceeds some threshold. However, a threshold which is high enough to rule out irrelevant attributes, will also rule out relevant ones.

   An alternative is the χ^2 test: if attribute A is *irrelevant* (statistically independent) to the class of an object in S, the expected value p'_i of p_i (the number of elements of class P is S) would be

   $$p' = p\frac{p_i + n_i}{p+n} = p\frac{|S_i|}{|S|}$$

   if n'_i is the corresponding expected value of n_i, the statistic

   $$\Sigma_{i=1}^{v} \frac{(p_i - p'_i)^2}{p} + \frac{(n_i - n'_i)^2}{n'_i}$$

   is approximately χ^2 with $v - 1$ degrees of freedom. The tree construction procedure is modified such that only attributes whose irrelevance can be rejected with a very high confidence level will be used.

8. Missing attribute values: The distribution of unknown values is assumed to be in proportion to the relative frequency of these values in S. If branching on one of the missing attribute values is required, ID3 estimates the probability (the product of the relative frequencies of the chosen values for the unknown attributes) that this is the correct choice of values for the attributes. The result of the classification is the class with the highest probability.

9. Windows: Use a randomly chosen subset (window) as a training set to generate a tree. As long as not all examples are correctly classified, some of the incorrectly classified examples are added to the window to generate a new tree. Experimental results show that correct decision trees are found within a few iterations.

10. Incremental learning: ID5R does not create a new tree for each new example, but instead restructures the tree to make it consistent with the current and all previous example.

11. Conclusion: The system performs very well on a wide range of application domains, such as medical, chess-end games, etc. The classification accuracy is high. But the system does not use domain knowledge, and trees are not easy to understand (but transformable to decision rules).

7.2 AQ15

1. AQ15: an inductive learning system that generates decision rules. Construction of strong rules: covers all positive examples but no negative ones. Attributes can be of 3 types: nominal, linear or (hierarchically) structured.

2. Search space: Rules are represented in the VL_1 (Variable-valued Logic system 1) notation: *selector*: 'color = red ∨ green'. *complex* (set-description): a conjunction of selectors. *cover*: the conditional part of a decision rule, formed of a disjunction of complexes.

   AQ is to find the most preferred strong rules. This preference criterion is defined by the user, to reflect the needs of the application domain.

   Operations: *addition* of a complex to the cover of a rule, and *intersection* of a set of complexes with a set of selectors.

3. Search algorithm:

   while partial cover does not cover all positive examples **do**

   (a) Select a seed (an uncovered positive example)

   (b) Generate a star (maximally general complexes covering the seed and no negative examples), which is implemented as follows:

       while partial cover covers negative examples **do**

       i. select a covered negative example.

       ii. generate a *partial star*: all maximal general selectors that cover the seed and exclude the negative example.

       iii. generate a new partial star by intersecting the current partial star with the partial star constructed so far.

       iv. trim the partial star, i.e., retain only the *maxstar* best complexes.

   (c) Select the best complex, as defined by quality function.

   (d) Add the complex to the cover.

4. Inconsistent data and noise: AQ provides 3 options: treats inconsistent examples either as positive examples, negative examples, or neglect them. If the statistical information about the probabilities of inconsistent examples is available, preclassify them according to the maximum likelihood.

5. Constructive induction: The AQ system uses domain knowledge, expressed in the form of rules (logic rules and arithmetic rules), to generate new attributes. Logic rules represent background class definitions, constraints among classes, generalization hierarchies, causal dependencies, etc.

6. Incremental learning: AQ15 has incremental learning facility and implements *learning with full memory*: remember all examples that were seen so far, as well as the rules it formed.

7. Conclusion: Discover strong rules. Inconsistent/noisy data are handled by preprocessing (inconsistent examples) and postprocessing (rule truncation). Tested in several domains.

7.3 CN2

1. CN2 (Clark and Niblett): To incorporate noise handling technique (e.g. search space pruning) in the learning (e.g. AQ) algorithm itself. It combines the best features of both ID3 and AQ (It uses the pruning techniques similar to those in ID3 and the conditional rules in AQ).

2. Search space: Output an ordered set of "if $\langle$complex$\rangle$then $\langle$class$\rangle$". During the search, complexes are specialized by adding a conjunctive selector or by removing a disjunctive value in one of the selectors.

   Quality criterion: accurate and significant. Accuracy: Finding the set E of examples that are covered by the complex, and the probability distribution $P = (p_1, p_2, \ldots, p_n)$ of examples in E among the n classes.

   Quality: the lower the entropy, the better the quality (prefers complexes covering a large number of examples of a single class and few examples of other classes).

   $$Entropy = -\Sigma_{i=1}^{n} p_i \log_2(p_i).$$

   The significance of the complex (locate a regularity unlikely to have occurred by chance, and thus reflect a genuine correlation between attribute values and classes). The likelihood ratio statistic:

   $$2\Sigma_{i=1}^{n} f_i \log(f_i/e_i)$$

   where the distribution $F = (f_1, \ldots, f_n)$ is the observed frequency distribution, and $E = (e_1, \ldots, e_n)$ is the expected frequency distribution. Under suitable assumptions, this statistic is distributed approximately as χ^2 with $n - 1$ degrees of freedom. The lower the score, the more likely that the apparent regularity is due to chance. Only complexes that pass a user defined minimum threshold of significance are taken into consideration.

3. Search algorithm: generates a decision list in an iterative fashion, constructing a rule at each iteration. A rule is constructed by searching for a complex that covers a large number of examples of an arbitrary class C_i and few of other classes. Having found a good example, the algorithm removes those examples it covers from the training set and adds the rule '*if* $\langle$*complex*$\rangle$*then predict* C_i' to the end of the decision list. Repeat the process for the remaining set until no further complexes of sufficient quality are found.

7.4 DBLearn

1. A reasonably good summary of our work.

2. For more detail, see our several papers, e.g., Cai, Cercone & Han (1991, *Knowledge Discovery in Databases*), Han, Cai & Cercone (1992, *VLDB*), Han, Cai & Cercone, (1993, *IEEE Trans. Knowledge & Data Engineering*), Han, Nishio & Kawano (1993, *KB&KS'93*), Han & Fu (1994, *KDD'94*).

7.5 Meta-Dendral

1. A special purpose machine learning system, designed for the automatic discovery of rules of mass spectroscopy.

7.6 RADIX/RX

1. Discovery of relationships in a clinical database. The notion of *time* is included: the generated knowledge consists of *causal* relationship.

2. Data representation: $P \times A \times T$, where P patients, A attributes and T different moments.

3. Knowledge representation:

   Correlation: A is correlated with B, if an example has property A, it also has property B, and vice versa.

   Time precedence: A is generally followed by B, i.e., A is the cause, and B is the effect.

   Nonspuriousness: A and B are correlated if there is no known third property (confounding property) C responsible for the correlation.

   Each arc is labeled to denote some extra information, such as intensity, frequency, direction, and validity.

4. RX does not use domain knowledge to guide the search for interesting relationships. But Radix, a new version, uses medical knowledge to focus discovery.

8 Numerical and Hybrid Learning Systems

Discretization of numerical values: maps numerical values into nonnumerical, finite number of symbols, but involves loss of information.

8.1 Bacon

1. The Bacon system uses data analysis algorithms to discover mathematical relationships in numerical data.

2. Search space: The preference criterion — A term is correct if it has a constant value for all examples (e.g., discovery of Kepler's 3rd law).

   Operations that transform terms in other terms include:

   (a) Numerical operations, such as multiplication or division of the term with a variable.
   (b) Specialization of the term: restricts the set of examples to a particular subset of the data.
   (c) Creating terms for the slope and intercept of linear relations between two variables.
   (d) Creating terms for constant modulo-n relations.

3. Search algorithm: Top-down search: generating an algorithm, and checking the correctness of this term for the set of examples.

   Heuristics, e.g., if the values for the term and a previously constructed term increase monotonously, the ratio of the term and the variable is taken; or if two terms vary inversely, the product is taken.

4. Intrinsic properties: Sometimes the set of examples consists of tests for different entities, instead of test on a single entity. Each example is then labeled with its entity. If Bacon cannot construct a constant term for the entire set, but only a term which is constant for each entity, it assumes that these entities have some intrinsic property. E.g., $V = IR$ for different batteries.

5. Conclusion. Drawbacks: It assumes noise-free, complete data. An enhancement could be to use a more sophisticated preference criterion, e.g., the smallest squares method, to define the matching between a term and a set of numerical data. Other problems: assume *all* variables are relevant, and the efficiency problem: use more sophisticated heuristics?

8.2 KEDS

1. KEDS (Knowledge-base Equation Discovery System): [divide and conquer] breaks the problem space into smaller regions and constructs functions for each of these regions.

2. Search space: generate rules called region-equation pairs

$$R_i \Rightarrow y = f_i(x_1, x_2, \ldots),$$

   where R_i is the description of a region, i.e., a condition on attribute values, y is the predicted variable, and $x_1, x_2, \ldots$ are predicting variables. Regions are hyperrectangles, hence their description consists of intervals for each variable.

   A function "y =$?ax_1^2$+$?bx_2$+$?c$" is called a template (a function with unknown coefficients). The user defines a set of templates (expectations about the possible forms of the numerical relationships). The equation f_i is an instantiation of one of the user-defined templates. The model is a set of region-equation pairs, such that the regions form a partitioning of the training set.

3. Search algorithm: A two-phase (discovery and partitioning) top-down search process. Parameters to control the search: majority trend factor m and the error bound ϵ.

   In the discovery phase, an equation is constructed that predicts part of the examples. First, a template is chosen, and examples are randomly chosen to compute the unknown coefficients in the template. The instantiated template – the equation f_i – has a cover, i.e., the set of examples whose response attributes are predicted with at most ϵ error by the equation.

   If the fraction of the training set covered by the equation is greater than m, then the training set is partitioned into positive and negative examples, which are used to determine the region in the second phase (the partitioning phase). A conventional supervised learning technique (a continuous version of the AQ algorithm) is used to find the description R_i of the covered region.

   The search process is iterative: the discovery phase can be invoked to refine the equation, using only examples from the region. The refined equation is again passed to the partitioning phase, ..., until the region-equation pair stops improving (either in accuracy or in covering more data points).

4. Summary: Region-equation pairs are a more powerful knowledge representation than purely numerical functions. The learning speed is quite low, even on small data sets – need more heuristics to guide the search.

9 Conclusions and Future Research

9.1 Data mining for beginners

1. Defining the mining task: which relationships to be discovered, which characteristics of objects, the knowledge structure, which information (predicting variables) can be used for this prediction.

2. Selecting data: relevant set of data.

3. Knowledge representation: variants of relational algebra selection operator in the propositional-like representations, e.g., decision trees (ID3), production rules (AQ, DBLearn), or decision lists (CN2).

   It is useful to employ a single representation formalism for both the knowledge structures and the domain knowledge. This representation has to be sufficiently expressive, e.g., first-order logic.

4. Transformation operators: Operators that transform one knowledge structure into another should be chosen such that they align as much as possible with the quality function.

5. Quality function: criterion – classification accuracy and understandability for humans (preferring simple structure).

6. Search algorithm: Bottom-up vs. top-down. Exhaustive search vs. beam search vs. hill climber.

System	*Search strategy*		*Extensions estimated*
ID3	top-down	hill climber	all
AQ15	top-down	beam search	some
CN2	top-down	beam search	all
DBLearn	bottom-up	hill climber	all
Meta-Dendral	top-down	exhaustive	–
RX	top-down	exhaustive (constrained)	–
Bacon	top-down	hill climber	none
KEDS (discovery)	top-down	exhaustive	–
KEDS (partitioning)	top-down	beam search	some

Table 1: KDD systems and their search strategies.

7. Heuristics: Use heuristics to choose the next operation. Heuristics: estimate the improvements resulting from the application of operations to the current knowledge structure, using the quality function or another cheaper criterion.

8. Noise and missing information: Use some kind of pruning to deal with noise. For incomplete information, a data mine system can either preprocess the training set (remove examples with missing values or replace the missing value with the most likely one) or the algorithm can be adapted to handle missing value itself (e.g., ID3).

9.2 Research topics

1. Machine learning: knowledge representation and search strategies. Most ML systems search for deterministic relationships, whereas most relationships in data mining are probabilistic.

2. Statistical techniques: A natural choice, may outperform others in speed.

3. Intelligent database interfaces: windows, optimization techniques, efficient caching algorithms, effective search strategies and heuristics, user-interaction, representation of domain knowledge and its applications in the search algorithm.

9.3 Future directions

1. More kinds of rules, aggregates, search for trends, etc.

2. Supervised vs. unsupervised learning.

3. 3 scenarios: (1) application of ML systems to current applications, such as databases; (2) multiple strategy learning (different algorithms for different application domains); and (3) enhancing conventional systems (e.g., editors, DBMS, user interface) by incorporating learning capabilities.