Chapter 1

Rule-based Classification

Xiao-Li Li
Institute for Infocomm Research
Singapore
xlli@i2r.a-star.edu.sg

Bing Liu
University of Illinois at Chicago (UIC)
Chicago, IL
liub@cs.uic.edu

1.1 Introduction

Classification is an important problem in machine learning and data mining. It has been widely applied in many real-world applications. Traditionally, to build a classifier, a user first needs to collect a set of training examples/instances that are labeled with predefined classes. A classification algorithm is then applied to the training data to build a classifier that is subsequently employed to assign the predefined classes to test instances (for evaluation) or future instances (for application) [1].

In the past three decades, many classification techniques, such as Support Vector Machines (SVM) [2], Neural Network (NN) [3], Rule Learning [9], Naïve Bayesian (NB) [5], K-Nearest Neighbour (KNN) [6], Decision Tree [4], have been proposed. In this chapter, we focus on rule learning, also called rule-based classification. Rule learning is valuable due to the following advantages.

1. Rules are very natural for knowledge representation, as people can understand and interpret them easily.

2. Classification results are easy to explain. Based on a rule database and input data from the user, we can explain which rule or set of rules is used to infer the class label so that the user is clear about the logic behind the inference.

3. Rule-based classification models can be easily enhanced and complemented by adding new rules from domain experts based on their domain knowledge. This has been successfully implemented in many expert systems.

4. Once rules are learned and stored into a rule database, we can subsequently use them to classify new instances rapidly through building index structures for rules and searching for relevant rules efficiently.

5. Rule based classification systems are competitive with other classification algorithms and in many cases are even better than them.

Now, let us have more detailed discussions about rules. Clearly, rules can represent information or knowledge in a very simple and effective way. They provide a very good data model that human beings can understand very well. Rules are represented in the logic form as **IF-THEN** statements, e.g. a commonly used rule can be expressed as follows:

IF *condition* **THEN** *conclusion.*

where the **IF** part is called the "antecedent" or "condition" and the **THEN** part is called the "consequent" or "conclusion" . It basically means that if the *condition* of the rule is satisfied, we can infer or deduct the *conclusion.* As such, we can also write the rule in the following format, namely, *condition* $\rightarrow$ *conclusion.* The *condition* typically consists of one or more feature tests (e.g. $feature_1 > value_2$, $feature_5 = value_3$) connected using logic operators (i.e. "and", "or", "not"). For example, we can have a rule like: If *sex*="*female*" *and* (35 < *age* <45) *and* (*salary*="*high*" or *creditlimit*="*high*"), then *potential_customer* ="*yes*". In the context of classification, the conclusion can be the class label, e.g. "*yes*" (*potential_customer* ="*yes*") or "*no*" (*potential_customer* ="*no*"). In other words, rules can be used for classification if its "consequent" will be one of those predefined classes and its "antecedent" or "precondition" contains conditions of various features and their corresponding values.

Many machine learning and data mining techniques have been proposed to automatically learn rules from data. In computer science domain, rule-based systems have been extensively used as an effective way to store knowledge and to do logic inference. Furthermore, based on the given inputs and the rule database, we can manipulate the stored knowledge for interpreting the generated outputs as well as for decision making. Particularly, rules and rule based classification systems have been widely applied in various expert systems, such as fault diagnosis for aerospace and manufacturing, medical diagnosis, highly interactive or conversational Q&A system, mortgage expert systems etc.

In this chapter, we will introduce some representative techniques for rule-based classification, which includes two key components, namely 1) rule induction which learns rules from a given training database/set automatically; and 2) classification, which makes use of the learned rule set for classification. Particularly, we will study two popular rule-based classification approaches: (1) rule induction and (2) classification based on association rule mining.

1. **Rule induction**. Many rule induction/learning algorithms, such as [9], [10], [11], [12], [13], [14], have adopted the sequential covering strategy, whose basic idea is to learn a list of rules from the training data *sequentially*, or one by one. That is, once a new rule has been learned, it will remove the corresponding training examples that it *covers*, i.e. remove those training examples that satisfy the rule antecedent. This learning process, i.e. learn a new rule and remove its covered training data, is repeated until rules can cover the whole training data or no new rule can be learned from the remaining training data.

2. **Classification based on association rule mining**. Association rule mining [16], is perhaps the most important model invented by data mining researchers. Many efficient algorithms have been proposed to detect association rules from large amount of data. One special type of association rules is called class association rules (CARs). The consequent of a CAR must be a class label, which makes it attractive for classification purposes. We will describe Classification Based on Associations (CBA) — the first system that uses association rules for classification [30], as well as a number of more recent algorithms that perform classification based on mining and applying association rules.

1.2 Rule Induction

The process of learning rules from data directly is called rule induction or rule learning. Most rule induction systems have utilized a learning strategy which is described as *sequential covering*. A rule-based classifier built with this strategy typically consists of a list of discovered rules, which is also called a *decision list* [8]. Note in the *decision list*, the ordering of the rules is very important since it decides which rule should be first used for classification.

The basic idea of sequential covering is to learn a list of rules *sequentially*, one at a time, to cover the whole training data. A rule covering a data instance (either training or test instance), means that the instance satisfies the conditions of the rule. As such, covering the whole training data simply means every training instance in the training data satisfies the conditions of at least one rule in the *decision list* — it is possible that one training/test instance satisfies multiple rules and typically each rule can cover multiple instances. A key learning step using the *sequential covering* strategy is as follows: after each rule is learned, the training examples covered by the rule are removed from the training data and only the remaining training data are used to learn subsequent rules. These key learning steps are performed repeatedly until the remaining training set becomes empty or no new rule can be learned from the training data.

In this section, we study two specific algorithms based on the sequential covering strategy. Both of them are well-known and highly cited. The first algorithm is the CN2 induction algorithm [9] and the second algorithm is based on the ideas from RIPPER algorithm and its variations such as RIPPER [13], FOIL [10], I-REP [11], and REP [12]. Note some ideas are also taken from [14].

1.2.1 Two algorithms for Rule Induction

We now present these two algorithms, namely CN2 and RIPPER (and its variations), in section 1.2.1.1 and section 1.2.1.2 respectively.

1.2.1.1 CN2 Induction Algorithm (Ordered Rules)

CN2 algorithm learns each rule without pre-fixing a class [9]. That is, in each iteration, a rule for any class may be learned. As such, rules for different classes may intermix in the final *decision list* $RULE\_LIST$. As we have mentioned earlier, the ordering of rules is essential for classification, as rules are highly dependent on each other.

CN2 was designed to incorporate ideas from both the AQ algorithm [18] and the ID3 algorithm [4]. Before presenting CN2, we first introduce several basic concepts that were introduced in the AQ algorithm as well as later in CN2 algorithm. Each rule is in the form of "if $< cover >$ then predict $< class >$", where $< cover >$ is a Boolean combination of multiple attribute tests. The basic test on an attribute is called a *selector*, e.g. $< cloudy = yes >$, $< weather = wet \vee stormy >$, and $< Temperature \geq 25 >$. A conjunction of selectors is called a *complex*, e.g. $< cloudy = yes >$ *and* $< weather = wet \vee stormy >$. A disjunct of multiple complexes is called a *cover*.

The CN2 rule induction algorithm, which is based on ordered rules, is given below, which uses sequential covering.

INPUT

Let E be a set of classified (training) examples

Let $SELECTORS$ be the set of all possible selectors

CN2 Induction Algorithm CN2(E)

```
1. Let $RULE\_LIST$ be the empty list;    // initialize an empty rule set in the beginning
2. Repeat until $Best\_CPX$ is nil or $E$ is empty;
3.     Let $Best\_CPX$ be $Find\_Best\_Complex(E)$
4.     If $Best\_CPX$ is not nil
5.     Then let $E'$ be the examples covered by $Best\_CPX$
6.              Remove from $E$ the examples $E'$ covered by $Best\_CPX$
7.              Let $C$ be the most common class of examples in $E'$
8.            Add the rule "If $Best\_CPX$ then the class is $C$" to the end of $RULE\_LIST$.
9. **Output** $RULE\_LIST$.
```

In this algorithm, we need two inputs, namely, E and $SELECTORS$. E is the training data and $SELECTORS$ is the set of all possible selectors that test each attribute and its corresponding values. Set $RULE\_LIST$ is the *decision list*, storing the final output list of rules, which is initialized as to empty set in step 1. $Best\_CPX$ records the best rule detected in each iteration. The function $Find\_Best\_Complex(E)$ learns the $Best\_CPX$. We will elaborate the details of this function later in section 1.2.2. Steps 2 to 8 form a Repeat-loop which learns the best rule and refine the training data. In particular, in each Repeat-loop, once a non-empty rule is learned from the data (steps 3 and 4), all the training examples that are covered by the rule are removed from the data (steps 5 and 6). The rule discovered, consisting of the rule condition and the most common class label of the examples covered by the rule, is added at the end of $RULE\_LIST$ (steps 7 and 8).

The stopping criteria for the Repeat-loop (from steps 2–8) can be either $E = \emptyset$ (no training examples left for learning) or Rule $Best\_CPX$ is *nil* (there is no new rule learned from the training data). After the rule learning process completes (i.e. satisfies one of the two stopping criteria), a default class c is inserted at the end of $RULE\_LIST$. This step is performed because of the following two reasons: 1) there may still be some training examples that are not covered by any rule as no good rule can be mined from them, and 2) some test instances may not be covered by any rule in the $RULE\_LIST$ and thus we cannot classify it if we do not have a *default-class*. Clearly, with this *default-class*, we are able to classify any test instance. The *default-class* is typically the majority class among all the classes in the training data, which will be used only if no rule learned from the training data can be used to classify a test example. The final list of rules, together with the *default-class*, is represented as follows:

$<r_1, r_2, \ldots, r_k$, *default-class*$>$, where r_i is a rule mined from the training data.

Finally, using the list of rules for classification is rather straightforward. For a given test instance, we simply try each rule in the list *sequentially*, starting from r_1, then r_2 (if r_1 cannot cover the test instance), r_3 (if both r_1 and r_2 cannot cover the test instance) and so on. The class consequent of the first rule that covers this test instance is assigned as the class of the test instance. If no rule (from $r_1, r_2, \ldots, r_k$) applies to the test instance, the *default-class* is applied.

1.2.1.2 RIPPER Algorithm and its Variations (Ordered Classes)

We now introduce the second algorithm (Ordered Classes), which is based on the RIPPER algorithm [13] [51] [71], as well as earlier variations such as FOIL [10], I-REP [11], and REP [12].

RIPPER algorithm and its variations (D, C)

```
1.  RuleList ← ∅;    // initialize RuleList as an empty rule set
2.  For each class $c \in C$ do
3.      Prepare data $(Pos, Neg)$, where $Pos$ contains all the examples of class $c$ from $D$,
    and $Neg$ contains the rest of the examples in $D$;
4.      While $Pos \neq \emptyset$ do
5.          Rule ← learn-one-rule$(Pos, Neg, c)$
6.          If Rule is NULL then
7.              exit-While-loop
8.          Else RuleList ← insert Rule at the end of RuleList;
9.              Remove examples covered by Rule from $(Pos, Neg)$
10.         EndIf
11.     EndWhile
12. EndFor
13. Output RuleList.
```

Different from the CN2 algorithm which learns each rule without pre-fixing a class, RIPPLE learns all rules for each class individually. In particular, only after rule learning for one class is completed, it moves on to the next class. As such, all rules for each class appear together in the rule *decision list*. The sequence of rules for each individual class is not important, but the rule subsets for different classes are ordered and still important. The algorithm usually mines rules for the least frequent/minority/rare class first, then the second minority class, and so on. This process ensures that some rules are learned for rare or minority classes. Otherwise, they may be dominated by frequent or majority classes and we will end up with no rules for the minority classes. The RIPPER rule induction algorithm is shown as follows, which is also based on sequential covering:

In this algorithm, the data set D is split into two subsets, namely, Pos and Neg, where Pos contains all the examples of class c from D, and Neg the rest of the examples in D (see step 3), i.e. in a one-vs-others manner. Here $c \in C$ is the current working class of the algorithm, which is initialized as the least frequent class in the first iteration. As we can observe from the algorithm, steps 2 to 12 is a **For-loop** which goes through all the classes one by one, starting from the minority class. That is why this method is called *Ordered Classes*, from the least frequent class to the most frequent class. For each class c, we have an internal **While-loop** from steps 4 to 11 which includes the rule learning procedure, i.e., perform the Learn-One-Rule() function to learn a rule *Rule* in step 5; insert the learned *Rule* at the end of *RuleList* in step 8; remove examples covered by *Rule* from (Pos, Neg) in step 9. Note two stopping conditions for internal rule learning of each class c are in given in step 4 and step 6 respectively — we stop the **while-loop** for the internal rule learning process for the class c when the Pos becomes empty or no new rule can be learned by function Learn-One-Rule(Pos, Neg, c) from the remaining training data.

The other parts of the algorithm are very similar to those of the CN2 algorithm. The Learn-One-Rule() function will be described later in section 1.2.2.

Finally, applying the *RuleList* for classification is done in a similar way as for CN2 algorithm. The only difference is that the order of all rules within each class is not important anymore since they share the same class label which will lead to the same classification results. Since the rules are now ranked by classes, given a test example, we will try the rules for the least frequent classes first until we can find a single rule that can cover the test example to perform its classification; otherwise, we have to apply the *default-class*.

1.2.2 Learn One Rule in Rule Learning

In the two algorithms described above, we have not elaborated on the two important functions used by them, where the first algorithm uses $Find\_Best\_Complex()$ and the second algorithm uses learn-one-rule(). In this section, we explain the overall idea of the two functions that learn a rule from all or partial training data.

First, the rule starts with an empty set of conditions. In the first iteration, only one condition will be added. In order to find a best condition to add, all possible conditions are explored, which form a set of candidate rules. A condition is an attribute-value pair in the form of A_i *op* v, where A_i is an attribute and v is a value of A_i. Particularly, for a discrete attribute of v, assuming *op* is "=", then a condition will become $A_i = v$. On the other hand, for a continuous attribute, $op \in \{>, \leq\}$ and the condition becomes $A_i > v$ or $A_i \leq v$. The function then evaluates all the possible candidates to detect the *best* one from them (all the remaining candidates are discarded). Note the *best* candidate condition should be the one that can be used to better distinguish different classes, e.g. through an entropy function which has been successfully used in decision tree learning [4].

Next, after the first best condition is added, it further explores to add the second condition and so on in the same manner until some stopping conditions are satisfied. Note that we have omitted the rule class here because it implies the majority class of the training data *covered* by the conditions. In other words, it means that when we apply the rule, the class label that we predict should be correct most of the time.

Obviously, this is a heuristic and greedy algorithm in that after a condition is added, it will not be changed or removed through backtracking [15]. Ideally, we would like to explore all possible *combinations of attributes and values*. However, this is not practical as the number of possibilities grows exponentially with the increase number of conditions in rules. As such, in practice, the above greedy algorithm is used to perform rule induction efficiently.

Nevertheless, instead of keeping only the best set of conditions, we can improve the function a bit by keeping k best sets of conditions ($k > 1$) in each iteration. This is called the **beam search** (k beams), which ensures that a larger space (more attribute and value combinations) is explored, which may generate better results than the standard greedy algorithm which only keeps the best set of conditions.

Below, we present the two specific implementations of the functions, namely $Find\_Best\_Complex()$ and learn-one-rule() where $Find\_Best\_Complex()$ is used in the CN2 algorithm, and learn-one-rule() is used in the RIPPER algorithm and its variations.

$Find\_Best\_Complex(D)$

The function $Find\_Best\_Complex(D)$ uses beam search with k as its number of beams. The details of the function are given below.

Function $Find\_Best\_Complex(D)$

```
1.  $BestCond \leftarrow \emptyset$;    // rule with no condition.
2.  $candidateCondSet \leftarrow \{BestCond\}$;
3.  $attributeValuePairs \leftarrow$ the set of all attribute-value pairs in $D$ of the form $(A_i\ op\ v)$,
    where $A_i$ is an attribute and $v$ is a value or an interval;
4.  **While** $candidateCondSet \neq \emptyset$ **do**
5.      $newCandidateCondSet \leftarrow \emptyset$;
6.      **For** each candidate $cond$ in $candidateCondSet$ **do**
7.          **For** each attribute-value pair $a$ in $attributeValuePairs$ **do**
8.              $newCond \leftarrow cond \bigcup \{a\}$;
9.              $newCandidateCondSet \leftarrow newCandidateCondSet \bigcup \{newCond\}$;
10.         **EndFor**
11.     **EndFor**
12.     remove duplicates and inconsistencies, e.g., $\{A_i = v_1, A_i = v_2\}$;
13.     **For** each candidate $newCond$ in $newCandidateCondSet$ **do**
14.         **If** $evaluation(newCond, D) > evaluation(BestCond, D)$ **then**
15.             $BestCond \leftarrow newCond$
16.         **EndIf**
17.     **Endor**
18.     $candidateCondSet \leftarrow$ the $k$ best members of $newCandidateCondSet$ according to
    the results of the evaluation function;
19. **EndWhile**
20. **If** $evaluation(BestCond, D) - evaluation(\emptyset, D) > threshold$ **then**
21.     **Output** the rule: "$BestCond \rightarrow c$" where is $c$ the majority class of the data covered
    by $BestCond$
22. **Else Output** NULL
23. **EndIf**
```

In this function, set *BestCond* stores the conditions of a rule to be returned. The class is omitted here as it refers to the majority class of the training data covered by *BestCond*. Set *candidateCondSet* stores the current best condition sets (which are the frontier beam-s) and its size is less than or equal to k. Each condition set contains a set of conditions connected by "and" (conjunction). Set *newCandidateCondSet* stores all the new candidate condition sets after adding each attribute-value pair (a possible condition) to every candidate in *candidateCondSet* (steps 5-11). Steps 13-17 update the *BestCond*. Note that an evaluation function is used to assess whether each new candidate condition set is better than the existing best condition set *BestCond* (step 14). If a new candidate condition set has been found better, it then replaces the current *BestCond* (step 15). Step 18 updates *candidateCondSet*, which selects k best condition sets (new beams).

Once the final *BestCond* is found, it is evaluated to check if it is significantly better than without any condition ($\emptyset$) using a threshold (step 20). If yes, a rule is formed using *BestCond* and the most frequent (or the majority) class of the data covered by *BestCond* (step 21). If not, NULL is returned, indicating that no significant rule is found.

Note the evaluation() function shown below employs the entropy function, the same as in the decision tree learning, to evaluate how good the *BestCond* is.

```
Function evaluation($BestCond, D$)
    1. $D' \leftarrow$ the subset of training examples in $D$ covered by $BestCond$
    2. $entropy(D') = -\sum_{j=1}^{|C|} Pr(c_j) \times log_2 Pr(c_j)$;
    3. Output $-entropy(D')$    // since entropy measures impurity.
```

Specifically, in the first step of the evaluation() function, it obtains an example set D' which consists of a subset of training examples in D covered by *BestCond*. In its second step, it calculates an entropy function $entropy(D')$ based on the probability distribution — $Pr(c_j)$ is the probability of class c_j in the data set D', which is defined as the number of examples of class c_j in D' divided by the total number of examples in D'. In the entropy computation, $0 \times log 0 = 0$. The unit of entropy is **bit**. We now provide some examples to help understand the entropy measure.

Assume the data set D' has only two classes, namely positive class (c_1=P) and negative class (c_2=N). Based on the following three different combinations of probability distributions, we can compute their entropy values as follows:

1. The data set D' has 50% positive examples (i.e. $Pr(P) = 0.5$) and 50% negative examples (i.e. $Pr(N) = 0.5$). Then, $entropy(D') = -0.5 \times log_2 0.5 - 0.5 \times log_2 0.5 = 1$.

2. The data set D' has 20% positive examples (i.e. $Pr(P) = 0.2$) and 80% negative examples (i.e. $Pr(N) = 0.8$). Then, $entropy(D') = -0.2 \times log_2 0.2 - 0.8 \times log_2 0.8 = 0.722$.

3. The data set D' has 100% positive examples (i.e. $Pr(P) = 1$) and no negative examples (i.e. $Pr(N) = 0$). Then, $entropy(D') = -1 \times log_2 1 - 0 \times log_2 0 = 0$.

From the three scenario shown above, we can observe that when the data class becomes purer and purer (e.g. all or most of examples belong to one individual class), the entropy value becomes smaller and smaller. As a matter of fact, it can be shown that for this binary case (only has positive and negative classes), when $Pr(P) = 0.5$ and $Pr(N) = 0.5$, the entropy has the maximum value, i.e., 1 bit. When all the data in D' belong to one class, the entropy has the minimum value, i.e., 0 bit. It is clear that the entropy measures the amount of impurity according to the data class distribution. Obviously, we would like to have a rule which has a low entroy or even 0 bit, since it means that the rule will lead to one major class and we are thus more confident to apply the rule for classification.

In addition to the entropy function, other evaluation functions can also be applied. Note that when $BestCond = \emptyset$, it covers every example in D, i.e. $D = D'$.

Learn-One-Rule

In the Learn-One-Rule() function, a rule is first generated and then subjected to a pruning process. This method starts by splitting the positive and negative training data *Pos* and *Neg*, into growing and pruning sets. The growing sets, *GrowPos* and *GrowNeg*, are used to generate a rule, called *BestRule*. The pruning sets, *PrunePos* and *PruneNeg* are used to prune the rule because *BestRule* may overfit the training data with too many conditions, which could lead to poor predictive performance on the unseen test data. Note that *PrunePos* and *PruneNeg* are actually validation sets, which are used to access the rule's generalization. If a rule has 50% error rate in the validation sets, then it does not generalize well and thus the function does not output it.

Function Learn-One-Rule(*Pos*, *Neg*, *class*)

```
1. split (Pos, Neg) into (GrowPos, GrowNeg) and (PrunePos, PruneNeg)
2. BestRule ← GrowRule(GrowPos, GrowNeg, class)    // grow a new rule
3. BestRule ← PruneRule(BestRule, PrunePos, PruneNeg)    // prune the rule
4. If the error rate of BestRule on (PrunePos, PruneNeg) exceeds 50% Then
5.     return NULL
6. Endif
7. Output BestRule
```

GrowRule() function: GrowRule() generates a rule (called *BestRule*) by repeatedly adding a condition to its condition set that maximizes an evaluation function until the rule covers only some positive examples in *GrowPos* but no negative examples in *GrowNeg*, i.e. 100% purity. This is basically the same as the Function $Find\_Best\_Complex(E)$, but without beam search (i.e., only the best rule is kept in each iteration). Let the current partially developed rule be R:

$$\text{R: } av_1, \ldots, av_k \rightarrow class$$

where each av_j (j=1, 2, ... k) in rule R is a condition (an attribute-value pair). By adding a new condition av_{k+1}, we obtain the rule R^+: $av_1, \ldots, av_k, av_{k+1} \rightarrow class$. The evaluation function for R^+ is the following **information gain** criterion (which is different from the gain function used in decision tree learning):

$$gain(R, R^+) = p_1 \times (log_2 \frac{p_1}{p_1 + n_1} - log_2 \frac{p_0}{p_0 + n_0}) \tag{1.1}$$

where p_0 (respectively, n_0) is the number of positive (or negative) examples covered by R in Pos (or Neg), and p_1 (or n_1) is the number of positive (or negative) examples covered by R^+ in Pos (or Neg). R^+ will be better than R if R^+ can cover more proportion of positive examples than R. The GrowRule() function simply returns the rule R^+ that maximizes the gain.

PruneRule() function: To prune a rule, we consider deleting every subset of conditions from the *BestRule*, and choose the deletion that maximizes:

$$v(BestRule, PrunePos, PruneNeg) = \frac{p - n}{p + n}, \tag{1.2}$$

where p (respectively n) is the number of examples in *PrunePos* (or *PruneNeg*) covered by the current rule (after a deletion).

1.3 Classification Based on Association Rule Mining

In the last section, we introduced how to mine rules through rule induction systems. In this section, we discuss *Classification Based on Association Rule Mining*, which makes use of the association rule mining techniques to mine association rules and subsequently to perform classification task by applying the discovered rules. Note that *Classification Based on Association Rule Mining* detects *all rules* in data that satisfy the user-specified minimum support (minsup) and minimum confidence (minconf) constraints while a rule induction system detects only *a subset of the rules* for classification. In many real-world applications, rules that are not found by a rule induction system may be of high value for enhancing the classification performance, or for other uses

The basic idea of *Classification Based on Association Rule Mining* is to first find strong correlations or associations between the frequent itemsets and class labels based on association rule mining techniques. These rules can be subsequently used for classification for test examples. Empirical evaluations have demonstrated that classification based on association rules are competitive with the state-of-the-art classification models, such as decision trees, navie Bayes, and rule induction algorithms.

In section 1.3.1, we will present the concepts of association rule mining and an algorithm to automatically detect rules from transaction data in an efficient way. Then, in section 1.3.2, we will introduce mining class association rules, where the class labels or *target attributes*) are on the right-hand side of the rules. Finally, in section 1.3.3, we describe some techniques for performing classification based on discovered association rules.

1.3.1 Association Rule Mining

Association rule mining, formulated by Agrawal et al. in 1993 [16], is perhaps the most important model invented and extensively studied by the database and data mining communities. Mining association rules is a fundamental and unique data mining task. It aims to discover all co-occurrence relationships (or associations, correlations) among data items, from very large data sets in an efficient way. The discovered associations can also be very useful in data clustering, classification, regression and many other data mining tasks.

Association rules represent an important class of regularities in data. Over the past t-wo decades, data mining researchers have proposed many efficient association rule mining algorithms, which have been applied across a wide range of real-world application domain-s, including business, finance, economy, manufacturing, aerospace, biology, and medicine etc. One interesting and successful example is Amazon book recommendation. Once association rules are detected automatically from the book purchasing history database, they can be applied to recommend users those relevant books based on other people/community purchasing experiences.

The classic application of association rule mining is the market basket data analysis, aiming to determine how items purchased by customers in a supermarket (or a store/shop) are associated or co-occurring together. For example, an association rule mined from a market basket data could be:

$$Bread \rightarrow Milk\ [\textit{support} = 10\%,\ \textit{confidence} = 80\%].$$

The rule basically means we can use $Bread$ to infer $Milk$ or those customers who buy $Bread$ also frequently buy $Milk$. However, it should read together with two important quality metrics, namely *support* and *confidence*. Particularly, the *support* of 10% for this rule means that 10% customers buy Bread and Milk together, or 10% of all the transactions under analysis show that Bread and Milk are purchased together. In addition, A *confidence* of 80% means that those who buy Bread also buy Milk 80% of the time. This rule indicates that item Bread and item Milk are closely associated. Note in this rule, these two metrics are actually used to measure the rule strength, which will be defined in section 1.3.1.1. Typically, association rules are considered interesting or useful if they satisfy two constraints, namely their *support* is larger than a **minimum support threshold** and their *confidence* is larger a **minimum confidence threshold**. Both thresholds are typically provided by users and good thresholds may need users to investigate the mining results and vary the threholds multiple times.

Clearly, this association rule mining model is very generic and can be used in many other applications. For example, in the context of the Web and text documents, it can be used to find word co-occurrence relationships and Web usage patterns. It can also be used to find frequent substructures such as subgraphs, subtrees, or sublattices etc [19], as long as these substructures are frequently occurred together in the given dataset.

Note standard association rule mining, however, does not consider the sequence or temporal order in which the items are purchased. Sequential pattern mining takes the sequential information into consideration. An example of a sequential pattern is "5% of customers buy bed first, then mattress and then pillows". The items are not purchased at the same time, but one after another. Such patterns are useful in Web usage mining for analyzing click streams in server logs [20].

1.3.1.1 Definitions of Association Rules, Support and Confidence

Now we are ready to formally define the problem of mining association rules. Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of **items**. In the market basket data analysis scenario, for example, set I contains all the items sold in a supermarket. Let $T = (t_1, t_2, \ldots, t_n)$ be a set of **transactions** (the database), where each transaction t_i is a record, consisting of a set of items such that $t_i \subseteq I$. In other words, a transaction is simply a set of items purchased in a basket by a customer and a transaction database includes all the transactions which record all the baskets (or the purchasing history of all customers). An **association rule** is an implication of the following form: $X \to Y$, where $X \subset I$, $Y \subset I$, and $X \bigcap Y = \emptyset$. X (or Y) is a set of items, called an **itemset**.

Let us give a concrete example of a *transaction*: $t_i = \{Beef, Onion, Potato\}$, which indicates that a customer purchased three items, i.e. *Beef*, *Onion* and *Potato*, in his/her basket. An association rule could be in the following form:

$$Beef,\ Onion \to Potato,$$

where $\{Beef, Onion\}$ is X and $\{Potato\}$ is Y. Note brackets "{" and "}" are usually not explicitly included in both transactions and rules for simplicity.

As we mentioned before, each rule will be measured by its *support* and *confidence*. Next, we define both of them to evaluate the strength of rules.

A transaction $t_i \in T$ is said to contain an itemset X if X is a subset of t_i.

For example, itemset $\{Beef, Onion, Potato\}$ contains the following 7 itemsets: $\{Beef\}$, $\{Onion\}$, $\{Potato\}$, $\{Beef, Onion\}$, $\{Beef, Potato\}$, $\{Onion, Potato\}$, and $\{Beef, Onion, Potato\}$.

Below, we define the support of an itemset and a rule respectively.

Support of an itemset: The *support* count of an itemset X in T (denoted by $X.count$) is the number of transactions in T that contain X.

Support of a rule: The support of a *rule*, $X \to Y$ where X and Y are non-overlapping itemsets, is defined as the percentage of transactions in T that contains $X \cup Y$. The rule support thus determines how frequent the rule is applicable in the whole transaction set T.

TABLE 1.1: An example of a transaction database

t_1	Beef, Chicken, Milk
t_2	Beef, Cheese
t_3	Cheese, Boots
t_4	Beef, Chicken, Cheese
t_5	Beef, Chicken, Clothes, Cheese, Milk
t_6	Chicken, Clothes, Milk
t_7	Chicken, Milk, Clothes

Let n be the number of transactions in T. The support of the rule $X \rightarrow Y$ is computed as follows:

$$support = \frac{(X \cup Y).count}{n} \tag{1.3}$$

Note that support is a very important measure for filtering out those non-frequent rules that have a very low support since they occur in a very small percentage of the transactions and their occurrences could be simply due to chance.

Next, we define the confidence of a rule.

Confidence of a rule: The confidence of a rule, $X \rightarrow Y$, is the percentage of transactions in T that contain X also contain Y, which is computed as follows:

$$Confidence = \frac{(X \cup Y).count}{X.count} \tag{1.4}$$

Confidence thus determines the predictability and reliability of a rule. In other words, if the confidence of a rule is too low, then one cannot reliably infer or predict Y given X. Clearly, a rule with low predictability is not very useful in practice.

Given a transaction set T, the problem of mining association rules from T is to discover *all* association rules in T that have support and confidence greater than or equal to the user-specified **minimum support** (represented by **minsup**) and **minimum confidence** (represented by **minconf**).

Here we emphasize the keyword “all”, i.e., association rule mining requires the completeness of rules. The mining algorithms should not miss any rule that satisfies both **minsup** and **minconf** constraints.

Finally, we illustrate the concepts mentioned above using a concrete example, shown in the Table 1.1.

We are given a small transaction database, which contains a set of seven **transactions** $T = (t_1, t_2, \ldots, t_7)$. Each transaction t_i (i= 1, 2, ..., 7) is a set of items purchased in a basket in a supermarket by a customer. The set I is the set of all items sold in the supermarket, namely, $\{Beef, Boots, Cheese, Chicken, Clothes, Milk\}$.

Given two user-specified constraints, i.e. minsup = 30% and minconf = 80%, we aim to find all the association rules from the transaction database T. The following is one of association rules that we can obtain from T, where **sup**= 3/7 is the support of the rule, and **conf**= 3/3 is the confidence of the rule.

Chicken, Clothes $\rightarrow$ $Milk$ [**sup** = 3/7,**conf** = 3/3]

Let us now explain how to calculate the support and confidence for this transaction database. Out of the 7 transactions (i.e. n = 7 in equation 1.3), there are three of them, namely, t_5, t_6, t_7 contain itemset {Chicken, Clothes} $\cup$ {Milk} (i.e. $(X \cup Y).count$=3 in equation 1.3). As such, the support of the rule, **sup**=$(X \cup Y).count/n$=3/7=42.86%, which is larger than the minsup =30% (i.e. 42.86% > 30%).

On the other hand, out of the 3 transactions t_5, t_6, t_7 containing the condition itemset {Chicken, Clothes} (i.e. $X.count$=3), they also contain the consequent item {Milk}, i.e.{Chicken, Clothes} $\cup$ {Milk}= $(X \cup Y).count = 3$. As such, the confidence of the rule, **conf** $= (X \cup Y).count/X.count = 3/3 = 100\%$, which is larger than the minconf = 80% (100% > 80%). As this rule satisfies both the given minsup and minconf, it is thus valid.

We notice that there are potentially other valid rules. For example, the following one has two items as its consequent, i.e.

$$\text{Clothes} \rightarrow Milk, Chicken\ [\textbf{sup} = 3/7, \textbf{conf} = 3/3].$$

Over the past 20 years, a large number of association rule mining algorithms have been proposed. They mainly improve the mining efficiency since it is critical to have an efficient algorithm to deal with large scale transaction databases in many real-world applications. Please refer to [49] for detailed comparison across various algorithms in terms of their efficiencies.

Note that no matter which algorithms are applied, the final results, i.e. association rules minded, are all the same based on the definition of association rules. In other words, given a transaction data set T, as well as a minimum support **minsup** and a minimum confidence **minconf**, the set of association rules occurring in T is uniquely determined. All the algorithms should find the same set of rules although their computational efficiencies and memory requirements could be different. In the next session, we introduce the best known mining algorithm, namely the **Apriori** algorithm, proposed by Agrawal in [17].

1.3.1.2 The Introduction of Apriori Algorithm

The well-known Apriori algorithm consists of the following two steps:

1. *Generate all frequent itemsets*: A frequent itemset is an itemset that has a transaction support **sup** above **minsup**, i.e. **sup**>=**minsup**.

2. *Generate all confident association rules from the frequent itemsets*: A confident association rule is a rule with a confidence **conf** above **minconf**, i.e. **conf** >= **minconf**.

Note that the size of an itemset is defined as the number of items occurred in it — an itemset of size k (or k-itemset) contains k items. Following the example in Table 1.1, $\{Chicken, Clothes, Milk\}$ is a 3-itemset, containing 3 items, namely, $Chicken, Clothes$, and $Milk$. It is a *frequent 3-itemset* since its support **sup** = 3/7 is larger than **minsup** = 30%. From the 3-itemset, we can generate the following **three confident association rules** since their confidence **conf** = 100% are greater than **minconf** = 80%:

Rule 1: Chicken, Clothes $\rightarrow$ *Milk* [sup = 3/7, conf = 3/3]

Rule 2: Clothes, Milk $\rightarrow$ Chicken [sup = 3/7, conf = 3/3]

Rule 3: Clothes $\rightarrow$ Milk, Chicken [sup = 3/7, conf = 3/3].

Next, we discuss the two key steps of Apriori Algorithm, namely 1) Frequent Itemset Generation and 2) Association Rule Generation, in details.

STEP1: Frequent Itemset Generation

In the first step of Apriori algorithm, it generates all frequent itemsets efficiently by taking advantage of the following important property, i.e. *apriori property* or *downward closure property.*

Downward Closure Property: If an itemset has minimum support (or its support **sup** is larger than **minsup**), then its every non-empty subset also has minimum support.

The intuition behind this property is very simple because if a transaction contains a set of itemset X, then it must contain any non-empty subset of X. For example, $\{Chicken, Clothes, Milk\}$ is a frequent 3-itemset (sup=3/7). Any non-empty subset of $\{Chicken, Clothes, Milk\}$, say $\{Chicken, Clothes\}$ is also a frequent itemset since out of the 3 transactions containing $\{Chicken, Clothes, Milk\}$, they all contain its subset $\{Chicken, Clothes\}$. This property and a suitable *minsup* threshold have been exploited to prune a large number of itemsets that cannot be *frequent.* In the Apriori algorithm, it assumes that the items in I as well as all the itemsets are sorted in the **lexicographic order** to ensure efficient frequent itemset generation. For example, suppose we have a k-itemset $w = \{w_1, w_2, \ldots, w_k\}$ which consists of the items $w_1, w_2, \ldots, w_k$, where $w_1 < w_2 < \ldots < w_k$ according to the **lexicographic order**.

Apriori algorithm for frequent itemset generation [16] is a *bottom-up* based approach and uses level-wise search, which starts from 1-itemset and expand to higher level bigger itemsets, i.e. 2-itemset, 3-itemset and so on. The overall algorithm is shown in Algorithm 3. It generates all frequent itemsets by making multiple passes over the transaction database. In the first pass, it counts the supports of individual items, i.e. Level 1 items or 1-itemset in C_1 (as shown in step 1, C_1 is candidate 1-itemset) and determines whether each of them is frequent (step 2) where F_1 is the set of frequent 1-itemsets. After this initialization step, each of the subsequent pass k ($k \geq 2$), consists of the following three steps:

1. It starts with the seed set of itemsets F_{k-1} found to be frequent in the (k-1)-th pass. It then uses this seed set to generate candidate itemsets C_k (step 4), which are potential frequent itemsets. This step used the candidate-gen() procedure, as shown in Algorithm 4.

2. The transaction database is then passed over again and the actual support of each candidate itemset c in C_k is counted (steps 5-10). Note that it is not necessary to load the entire data into memory before processing. Instead, at any time point, only one transaction needs to reside in memory. This is a very important feature of the Apriori algorithm as it makes the algorithm scalable to huge data sets that cannot be loaded into memory.

3. At the end of the pass, it determines which of the candidate itemsets are actually frequent (step 11).

Algorithm 3: The Apriori algorithm for generating frequent itemsets

```
 1. $C_1 \leftarrow$ init-pass($T$); // the first pass over $T$
 2. $F_1 \leftarrow \{f | f \in C_1, f.count/n \geq minsup\}$; // $n$ is the no. of transactions in $T$;
 3. **For** ($k = 2$; $F_{k-1} \neq \emptyset$; $k$++) do
 4.     $C_k \leftarrow$ candidate-gen($F_{k-1}$);
 5.     **For** each transaction $t \in T$ do
 6.         **For** each candidate $c \in C_k$ do
 7.          **If** $c$ is contained in $t$ **then**
 8.             $c.count++$;
 9.         **EndFor**
10.     **EndFor**
11.     $F_k \leftarrow \{c \in C_k | c.count/n \geq minsup\}$
12. **EndFor**
13. **Output** $F \leftarrow \bigcup_k F_k$.
```

The final output of the algorithm is the set F of all frequent itemsets (step 13) where set F contains frequent itemsets with different sizes, i.e. frequent 1-itemsets, frequent 2-itemsets, ..., frequent k-itemsets (k is the highest order of the frequent itemsets).

Next, we elaborate the key candidate-gen() procedure which is called in step 4. Candidate-gen () generates candidate frequent itemsets in two steps, namely the *join step* and the *pruning step. Join step* (steps 2-6 in Algorithm 4): This step joins two frequent (k-1)-itemsets to produce a possible candidate c (step 6). The two frequent itemsets f_1 and f_2 have exactly the same $k-2$ items (i.e. $i_1, \ldots, i_{k-2}$) except the last one ($i_{k-1} \neq i'_{k-1}$ in steps 3-5). The joined k-itemset c is added to the set of candidates C_k (step 7). *Pruning step* (steps 8-11 in Algorithm 4): A candidate c from the join step may not be a final frequent item-set. This step determines whether all the k-1 subsets (there are k of them) of c are in F_{k-1}. If anyone of them is not in F_{k-1}, then c cannot be frequent according to the *downward closure property*, and is thus deleted from C_k.

Finally, we will provide an example to illustrate the candidate-gen() procedure.

Given a set of frequent itemsets at level 3, $F_3 = \{\{1,2,3\},\{1,2,4\},\{1,3,4\},\{1,3,5\},\{2,3,4\}\}$, the join step (which generates candidates C_4 for level 4) produces two candidate itemsets, $\{1,2,3,4\}$ and $\{1,3,4,5\}$. $\{1,2,3,4\}$ is generated by joining the first and the second itemsets in F_3 as their first and second items are the same $\{1,3,4,5\}$ is generated by joining the third and the fourth itemsets in F_3, i.e. $\{1,3,4\}$ and $\{1,3,5\}$. $\{1,3,4,5\}$ is prund because $\{3,4,5\}$ is not in F_3.

Procedure candidate-gen() is shown in the following Algorithm 4:

Algorithm 4: Candidate-gen(F_{k-1})

```
1.  $C_k = \emptyset$; // initialize the set of candidates
2.  **For** all $f_1, f_2 \in F_{k-1}$ // find all pairs of frequent itemsets
3.      with $f_1 = \{i_1, \ldots, i_{k-2}, i_{k-1}\}$ // that differ only in the last item
4.      with $f_2 = \{i_1, \ldots, i_{k-2}, i'_{k-1}\}$
5.      and $i_{k-1} < i'_{k-1}$ do // according to the lexicographic order
6.          $c \leftarrow \{i_1, \ldots, i_{k-1}, i'_{k-1}\}$; // join the two itemsets $f_1$ and $f_2$
7.          $C_k \leftarrow C_k \bigcup \{c\}$; // add the new itemset $c$ to the candidates
8.          **For** each ($k$ - 1)-subset $s$ of $c$ do
9.              **If** ($s \notin F_{k-1}$) **then**
10.                 delete $c$ from $C_k$; // delete $c$ from the candidates
11.         **EndFor**
12. **EndFor**
13. **Output** $C_k$.
```

We now provide a running example of the whole Apriori algorithm based on the transactions shown in Table 1.1. In this example, we have used **minsup** = 30%.

Apriori algorithm first scans the transaction data to count the supports of individual items. Those items whose supports are greater than or equal to 30% are regarded as frequent and are stored in set F_1, namely frequent 1-itemsets.

F_1: {{Beef}:4, {Cheese}:4, {Chicken}:5, {Clothes}:3, {Milk}:4}

In F_1, the number after each frequent itemset is the support count of the corresponding itemset. For example, {Beef}:4 means that the itemset {Beef} has occurred in 4 transactions, namely t_1, t_2, t_4, and t_5. A minimum support count of 3 is sufficient for being frequent (all the itemsets in F_1 have sufficient supports ≥ 3).

We then perform the Candidate-gen procedure using F_1, which generates the following candidate frequent itemsets C_2:

C_2: {{Beef, Cheese}, {Beef, Chicken}, {Beef, Clothes}, {Beef, Milk}, {Cheese, Chicken}, {Cheese, Clothes}, {Cheese, Milk}, {Chicken, Clothes}, {Chicken, Milk}, {Clothes, Milk}}

For each itemset in C_2, we need to determine if it is frequent by scanning the database again and storing the frequent 2-itemsets in set F_2:

F_2: {{Beef, Chicken}:3, {Beef, Cheese}:3, {Chicken, Clothes}:3, {Chicken, Milk}:4, {Clothes, Milk}:3}

We now complete the level-2 search (for all 2-itemsets). Similarly, we generate the candidate frequent itemsets C_3 via Candidate-gen procedure:

C_3: {{Chicken, Clothes, Milk}}

Note that in C_3, itemset {Beef, Cheese, Chicken}, is also produced in step 6 of the Candidate-gen procedure. However, as its subset {Cheese, Chicken} is not in F_2, it is pruned and not included in C_3, according to **downward closure property**.

Finally, we count the frequency of {Chicken, Clothes, Milk} in database and it is stored in F_3 given that its support is greater than the minimal support.

F_3: {{Chicken, Clothes, Milk}:3}.

Note that since we only have 1 itemset in F_3, the algorithm stops since we need at least 2 itemsets to generate a candidate itemset for C_4. Apriori algorithm is just a representative of a large number of association rule mining algorithms that have been developed over the 20 years. For more algorithms, please see [19].

STEP2: Association Rule Generation As we mention earlier, the Apriori algorithm can generate all frequent itemsets as well as all confident association rules. Interestingly, generating association rules is fairly straightforward compared with frequent itemset generation. In fact, we generate all association rules from frequent itemsets. For each frequent itemset f, we use all its non-empty subsets to generate association rules. In particular, for each such subset $\beta, \beta \subseteq f$, we output a rule 1.5 if the confidence condition in equation 1.6 is satisfied.

$$(f - \beta) \to \beta, \tag{1.5}$$

$$confidence = \frac{f.count}{(f - \beta).count} \geq minconf \tag{1.6}$$

Note the $f.count$ and $(f - \beta).count$ are the supports count of itemset f and itemset $(f - \beta)$ respectively. According to equation 1.3, our rule support is $f.count/n$, where n is the total number of transactions in the transaction set T. Clearly, if f is frequent, then any of its non-empty subsets is also frequent according to the downward closure property. In addition, all the support counts needed for confidence computation in equation 1.6, i.e. $f.count$ and $(f - \beta).count$, are available as we have recorded the supports for all the frequent itemsets during the mining process, e.g. using the Apriori algorithm. As such, there is no additional database scan needed for association rule generation.

1.3.2 Mining Class Association Rules

The association rules mined using Apriori algorithm are generic and flexible. An item can appear as part of the conditions or as part of the consequent in a rule. However, in some real-world applications, users are more interested in those rules with some fixed *target* items (or class labels) on the right-hand side. Obviously, such kind of rules are very useful for our rule-based classification models.

For example, banks typically maintain a customer database which contains demographic and financial information of individual customers (such as gender, age, ethnicity, address, employment status, salary, home ownership, current loan information, etc) as well as the target features such as whether or not they repaid the loans or defaulted. Using association rule mining technique, we can investigate what kind of customers are likely to repay (*good* credit risks) or to default (*bad* credit risks) — both of them are target feature values, so that banks can reduce the rate of loan defaults if they can predict those customers who are likely to default in advance based on their personal demographic and financial information. In other words, we are interested in a special set of rules whose consequents are only those target features — these rules are called class association rules (CARs) where we require only target feature values to occur as consequent of rules, although the conditions can be any items or their combinations from financial and demographic information.

Let T be a transaction data set consisting of n transactions. Each transaction in T has been labeled with a class y ($y \in Y$; Y is the set of all class labels or target features/items). Let I be the set of all items in T, and $I \bigcap Y = \emptyset$. Note here we treat the label set Y differently from the standard items in I and they do not have any overlapping. A class association rule (CAR) is an implication of the following form:

$$X \rightarrow y, where\ X \subseteq I,\ and\ y \in Y. \tag{1.7}$$

The definitions of **support** and **confidence** are the same as those for standard association rules. However, a class association rule is different from a standard association rule in the following two points:

1. The consequent of a CAR has only a *single* item, while the consequent of a standard association rule can have any number of items.

2. The consequent y of a CAR must be only from the class label set Y, i.e., $y \in Y$. No item from I can appear as the consequent, and no class label can appear as a rule condition. In contrast, a standard association rule can have any item as a condition or a consequent.

Clearly, the main objective of mining CARs is to automatically generate a complete set of CARs that satisfy both the user-specified minimum support constraint (minsup) and minimum confidence (minconf) constraint.

Intuitively, we can mine the given transaction data by first applying the Apriori algorithm to get all the rules and then perform a post-processing to select only those class association rules, as CARs are a *special* type of association rules with a target as its consequent. However, this is not efficient due to combinatorial explosion. Now, we present an efficient mining algorithm specifically designed for mining CARs.

This algorithm can mine CARs in a single step. The key operation is to find all *ruleitems* that have support above the given minsup. A ruleitem is a pair which has a *condset* and a class label y, namely, (*condset*, y), where *condset* $\subseteq I$ is a set of items, and $y \in Y$ is a class label. The support count of a *condset* (called *condsupCount*) is the number of transactions in T that contain the *condset.* The support count of a ruleitem (called *rulesupCount*) is the number of transactions in T that contain the *condset* and are associated with class y. Each ruleitem (*condset*, y) represents a rule:

$$condset \rightarrow y,$$

whose support is $(rulesupCount/n)$, where n is the total number of transactions in T, and whose confidence is $(rulesupCount/condsupCount)$.

Ruleitems that satisfy the minsup are called frequent ruleitems, while the rest are called infrequent ruleitems. Similarly, ruleitems that satisfy the minconf are called confident ruleitems and correspondingly the rules are confident.

The rule generation algorithm, called CAR-Apriori, is given in Algorithm 5. The CAR-Apriori algorithm is based on the Apriori algorithm, which generates all the frequent ruleitems by passing the database multiple times. In particular, it computes the support count in the first pass for each 1-ruleitem which contains only one item in its condset (step 1). All the 1-candidate ruleitems, which pair one item in I and a class label, are stored in set C_1.

$$C_1 = \{(\{i\}, y) | i \in I, and\ y \in Y\} \tag{1.8}$$

Then, step 2 chooses the *frequent* 1-ruleitems (and stores into F_1) whose support count is greater than or equal to the given minsup value. From frequent 1-ruleitems, we generate 1-condition CARs — rules with only one condition in step 3. In a subsequent pass, say k ($k \geq 2$), it starts with the seed set F_{k-1} of $(k-1)$ frequent ruleitems found in the $(k-1)$-th pass, and uses this seed set to generate new possibly frequent k-ruleitems, called candidate k-ruleitems (C_k in step 5). The actual support counts for both condsupCount and rulesupCount, are updated during the scan of the data (steps 6-13) for each candidate k-ruleitem. At the end of the data scan, it determines which of the candidate k-ruleitems in C_k are actually frequent (step 14). From the frequent k-ruleitems, step 15 generates k-condition CARs, i.e. class association rules with k conditions.

Algorithm 5: Algorithm CAR-Apriori(T)

```
1.  $C_1 \leftarrow$ init-pass($T$); // the first pass over $T$
2.  $F_1 \leftarrow \{f | f \in C_1, f.rulesupCount/f.condsupCount \geq minsup\}$; // $n$ is the no. of
    transactions in $T$;
3.  $CAR_1 \leftarrow \{f | f \in F_1, f.rulesupCount/n \geq minconf\}$; // $n$ is the no. of transactions
    in $T$;
4.  **For** (k = 2; $F_{k-1} \neq \emptyset$; k++) do
5.      $C_k \leftarrow$ CARcandidate-gen($F_{k-1}$);
6.      **For** each transaction $t \in T$ do
7.          **For** each candidate $c \in C_k$ do
8.           **If** $c.condset$ is contained in $t$ **then** // $c$ is a subset of $t$
9.                $c.condsupCount++$;
10.               **if** $t.class = c.class$ **then**
11.                 $c.rulesupCount++$;
12.         **EndFor**
13.     **EndFor**
14.     $F_k \leftarrow \{c \in C_k | c.rulesupCount/n \geq minsup\}$
15.     $CAR_k \leftarrow \{f | f \in F_k, f.rulesupCount/f.condsupCount \geq minconf\}$;
16. **EndFor**
17. **Output** $CAR \leftarrow \bigcup_k CAR_k$.
```

TABLE 1.2: A loan application data set

ID	Age	Has_job	Own_house	Credit_rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

One important observation regarding ruleitem generation is that if a ruleitem/rule has a confidence of 100%, then extending the ruleitem with more conditions, i.e. adding items to its condset, will also result in rules with 100% confidence although their supports may drop with additional items. In some applications, we may consider these subsequent rules with more conditions *redundant* because these additional conditions do not provide any more information for classification. As such, we should not extend such ruleitems in candidate generation for the next level (from $k-1$ to k), which can reduce the number of generated rules significantly. Of course, if desired, redundancy handling procedure can be added in the CAR-Apriori algorithm easily to stop the unnecessary expanding process.

Finally, the CARcandidate-gen() function is very similar to the candidate-gen() function in the Apriori algorithm, and it is thus not included here. The main difference lies in that in CARcandidate-gen(), ruleitems with the same class label are combined together by joining their condsets.

We now give an example to illustrate the usefulness of CARs. Table 1.2 shows a sample loan application dataset from a bank, which has four attributes, namely Age, Has_job, Own_house and Credit_rating. The first attribute Age has three possible values, i.e. young, middle and old. The second attribute Has_Job indicates whether an applicant has a job, with binary values: true (has a job) and false (does not have a job). The third attribute Own_house shows whether an applicant owns a house (similarly, it has true and false two values). The fourth attribute Credit_rating has three possible values: fair, good and excellent. The last column is the class/target attribute, which shows whether each loan application was approved (denoted by Yes) or not (denoted by No) by the bank.

Assume the user-specified minimal support *minsup* = 2/15 = 13.3% and the minimal confidence *minconf* = 70%, we can mine the above dataset to find the following rules that satisfy the two constraints:

$Own\_house = false, Has\_job = true \rightarrow Class = Yes$ [sup=3/15, conf=3/3]
$Own\_house = true \rightarrow Class = Yes$ [sup=6/15, conf=6/6]
$Own\_house = false, Has\_job = true \rightarrow Class = Yes$[sup=3/15, conf=3/3]
$Own\_house = false, Has\_job = false \rightarrow Class = No$ [sup=6/15, conf=6/6]

$Age = young, Has\_job = true \rightarrow Class = Yes$[sup=2/15, conf=2/2]
$Age = young, Has\_job = false \rightarrow Class = No$[sup=3/15, conf=3/3]
$Credit\_rating = fair \rightarrow Class = No$[sup=4/15, conf=4/5] .....

1.3.3 Classification Based on Associations

In this section, we discuss how to employ the discovered class association rules for classification purposes. Since the consequents of CARs are the class labels, it is thus logical to infer the class label of any test transaction, i.e., to do classification. CBA (Classification Based on Associations) is the first system that uses association rules for classification [30]. Note classifiers built using association rules are often called associative classifiers.

Following the above example, after we detect CARs, we intend to use them for learning a classification model to classify or automatically judge future loan applications. In other words, when a new customer visits the bank to apply for a loan, after providing his/her age, whether he/she has a job, whether he/she owns a house, and his/her credit rating, the classification model should predict whether his/her loan application should be approved so that we can use our constructed classification model to automate the loan application approval process.

1.3.3.1 Additional Discussion for CARs Mining

Before introducing how to build a classifier using CARs, we first give some additional discussions about some important points for mining high quality CARs.

Rule Pruning: CARs could be redundant and some of them are not statistically significant which make our classifier overfit the training examples and does not have good generalization capability. As such, we need to perform rule pruning to address these issues. Specifically, we can remove some conditions in CARs so that they are shorter, and have higher supports to be statistically significant. In addition, pruning some rules may cause some shorten/revised rules become redundant — we thus need to remove these repeated rules. Generally speaking, pruning rules could lead to a more concise and accurate rule set as shorter rules are less likely to overfit the training data and potentially perform well on the unseen test data. Pruning is also called generalization as it makes rules more general and more applicable to test instances. Of course, we still need to maintain high confidences of CARs during the pruning process so that we can achieve more reliable and accurate classification results once the confident rules are applied. Readers can refer to papers [31], [30] for details of some pruning methods.

Multiple Minimum Class Supports: in many real-life classification problems, the datasets could have uneven or imbalanced class distributions, where majority classes cover a large proportion of the training data, while other minority classes (rare or infrequent classes) only cover a very small portion of the training data. In such a scenario, a single minsup may be inadequate for mining CARs. For example, we have a fraud detection dataset with two classes C_1 (represents "normal class") and C_2 (denotes for "fraud class"). In this dataset, 99% of the data belong to the majority class C_1, and only 1% of the data belong to the minority class C_2, i.e. we do not have many instances from"fraud class". If we set minsup = 1.5%, we may not be able to find any rule for the minority class C_2 as this minsup is still too high for minority class C_2. To address the problem, we need to reduce the minsup, say set minsup = 0.2% so that we can detect some rules for class C_2. However, we may find a huge number of overfitting rules for the majority class C_1 because minsup = 0.2% is too low for class C_1. The solution for addressing this problem is to apply multiple minimum class supports for different classes, depending on their sizes. More specifically, we could assign a different minimum class support $minsup_i$ for each class C_i, i.e., all the rules of class C_i must satisfy corresponding $minsup_i$. Alternatively, we can provide one single total minsup, denoted by *total_minsup*, which is then distributed to each class according to the class distribution:

$$minsup_i = total\_minsup \times \frac{Number\ of\ Transactions\ in\ C_i}{Total\ Number\ of\ Transactions\ in\ Database} \quad (1.9)$$

The equation sets higher minsups for those majority classes while sets lower minsups for those minority classes.

Parameter Selection: The two parameters used in CARs mining are the minimum support and the minimum confidence. While different minimum confidences may also be used for each class, they do not affect the classification results much because the final classifier tends to use high confidence rules. As such, one minimum confidence is usually sufficient. We thus are mainly concern with how to determine the best support $minsup_i$ for each class C_i. Similar to other classification algorithms, we can apply the standard cross-validation technique to partition the training data into n folds where $n-1$ folds are used for training and the remaining 1 fold is used for testing (we can repeat this n times so that we have n different combinations of training and testing sets). Then, we can try different values for $minsup_i$ in the training data to mine CARs and finally choose the value for $minsup_i$ which gives the best average classification performance on the test sets.

1.3.3.2 Building a Classifier Using CARs

After all CARs are discovered through the mining algorithm, a classifier is built to exploit the rules for classification. We will introduce five kinds of approaches for classier building.

Use the Strongest Rule: This is perhaps the simplest strategy. It simply uses the most strongest/powerful CARs directly for classification. For each test instance, it first finds the strongest rule that covers the instance. Note that a rule covers an instance only if the instance satisfies the conditions of the rule. The class label of the strongest rule is then assigned to the test instance. The strength of a rule can be measured in various ways, e.g., based on rule confidence value only, χ^2 test, or a combination of both support and confidence values etc.

Select a Subset of the Rules to Build a Classifier: This method was used in the CBA system. This method is similar to the sequential covering method, but applied to class association rules with additional enhancements. Formally, let D and S be the training data set and the set of all discovered CARs respectively. The basic idea of this strategy is to select a subset L ($L \subseteq S$) of high confidence rules to cover the training data D. The set of selected rules, including a default class, is then used as the classifier. The selection of rules is based on a total order defined on the rules in S. Given two rules, r_i and r_j, we say $r_i \succ r_j$ or r_i precedes r_j or r_i has a higher precedence than r_j if

1. the confidence of r_i is greater than that of r_j, or
2. their confidences are the same, but the support of r_i is greater than that of r_j, or
3. both the confidences and supports of r_i and r_j are the same, but r_i is generated earlier than r_j.

A CBA classifier C is of the form:

$$C =< r_1, r_2, \ldots, r_k, default - class > \quad (1.10)$$

where $r_i \in S$, $r_i \succ r_j$ if $j > i$. When classifying a test case, the first rule that satisfy the case will be used to classify it. If there is not a single rule that can be applied to the test case, it takes the default class, i.e. $default - class$, in equation 1.10. A simplified version of the algorithm for building such a classifier is given in the following algorithm 6. The classifier is the $RuleList$.

Algorithm 6: Algorithm CBA (T)

```
1. $S = sort(S)$; // sorting is done according to the precedence $\succ$
2. $RuleList = \emptyset$ ; // the rule list classifier is initialized as empty set
3. **For** each rule $r \in S$ in sequence **do**
4.     **If** $(D \neq \emptyset)$ AND $r$ classifies at least one example in $D$ correctly **Then**
5.         delete from $D$ all training examples covered by $r$;
6.         add $r$ at the end of $RuleList$
7.     **EndIf**
8. **EndFor**
9. add the majority class as the default class at the end of RuleList
```

In algorithm 6, we first sort all the rules in S according to their precedence defined above. Then we then go through the rules one by one, from the highest precedence to the lowest precedence, during the for-loop. Particularly, for each rule, we will perform sequential covering from step 3 to 8. Finally, we construct our *RuleList* by appending the majority class so that we can classify any test instance.

Combine Multiple Rules: Like the first method *Use the Strongest Rule*, this method does not take any additional step to build a classifier. Instead, at the classification time, for each test instance, the system first searches a subset of rules that cover the instance.

1. If all the rules in the subset have the same class, then the class is assigned to the test instance.

2. If the rules have different classes, then the system divides the rules into a number of groups according to their classes, i.e., all rules of from same class are in the same group. The system then compares the aggregated effects of the rule groups and finds the strongest group. Finally, the class label of the strongest group is assigned to the test instance.

To measure the strength of each rule group, there again can be many possible ways. For example, the CMAR system uses a weighted χ^2 measure [31].

Class Association Rules as Features In this method, rules are used as features to augment the original data or simply form a new data set, which is subsequently fed to a traditional classification algorithm, e.g., Support Vector Machines (SVM), Decision Trees (DT), Naïve Bayesian (NB), K-Nearest Neighbour (KNN), etc.

To make use of CARs as features, only the conditional part of each rule is needed. For each training and test instance, we will construct a feature vector where each dimension corresponds to a specific rule. Specifically, if a training or test instance in the original data satisfies the conditional part of a rule, then the value of the feature/attribute in its vector will be assigned 1; 0 otherwise. The reason that this method is helpful is that CARs capture multi-attribute or multi-item correlations with class labels. Many classification algorithms, like Naïve Bayesian (which assumes the features are independent), do not take such correlations into consideration for classifier building. Clearly, the correlations among the features can provide additional insights on how different feature combinations can better infer the class label and thus they can be quite useful for classification. Several applications of this method have been reported [32], [33], [34], [35].

Classification Using Normal Association Rules

Not only can *class association rules* be used for classification, but also *normal association rules.* For example, normal association rules are regularly employed in e-commerce Web sites for product recommendations, which work as follows: When a customer purchases some products, the system will recommend him/her some other related products based on what he/she has already purchased as well as the previous transactions from all the customers.

Recommendation is essentially a classification or prediction problem. It predicts what a customer is likely to buy. Association rules are naturally applicable to such applications. The classification process consists of the following two steps:

1. The system first mines normal association rules using previous purchase transactions (the same as market basket transactions). Note, in this case, there are no fixed classes in the data and mined rules. Any item can appear on the left-hand side or the right-hand side of a rule. For recommendation purposes, usually only one item appears on the right-hand side of a rule.

2. At the prediction (or recommendation) stage, given a transaction (e.g., a set of items already purchased by a given customer), all the rules that cover the transaction are selected. The strongest rule is chosen and the item on the right-hand side of the rule (i.e., the consequent) is then the predicted item and is recommended to the user. If multiple rules are very strong, multiple items can be recommended to the user simultaneously.

This method is basically the same as the "use the strongest rule" method described above. Again, the rule strength can be measured in various ways, e.g., confidence, χ^2 test, or a combination of both support and confidence [42]. Clearly, the other methods, namely, *Select a Subset of the Rules to Build a Classifier*, and *Combine Multiple Rules*, can be applied as well.

The key advantage of using association rules for recommendation is that they can predict any item since any item can be the class item on the right-hand side.

Traditional classification algorithms, on the other hand, only work with a single fixed class attribute, and are not easily applicable to recommendations.

Finally, in recommendation system, multiple minimum supports can be of significant help. Otherwise, **rare items** will never be recommended, which causes the **coverage problem**. It is shown in [43] that using multiple minimum supports can dramatically increase the coverage. Note that rules from rule induction cannot be used for this recommendation purpose because each rule is not independent to each other.

1.3.4 Other Techniques for Association Rule Based Classification

Since CBA was proposed to use association rules for classification [30] in 1998, many techniques in this direction have been proposed. We introduce some of the representative ones, including CMAR [31], XRules [44]. Note XRules is specifically designed for classifying semi-structured data, such as XML.

1. CMAR

CMAR, stands for classification based on multiple association rules CMAR [31]. Like CBA, CMAR also consists of two phases, namely *rule generation phase* and *classification phase.* In *rule generation phase*, CMAR mines the complete set of rules in the form of $R : P \rightarrow c$, where P is a pattern in the transaction training data set, and c is a class label, i.e. R is a *class association rule.* The support and confidence of the rule R, namely

$sup(R)$ and $conf(R)$ satisfy the user pre-defined minimal support and confidence thresholds respectively.

CMAR used an effective and scalable association rule mining algorithm based on the FP-growth method [21]. As we know, existing association rule mining algorithms typically consist of two steps: 1) detect all the frequent patterns and 2) mine association rules that satisfy the confidence threshold based on the mined frequent patterns. CMAR, on the other hand, has no separated rule generation step. It constructs a class distribution-associated FP-tree and for every pattern it maintains the distribution of various class labels among examples matching the pattern, without any overhead in the procedure of counting database. As such, once a frequent pattern is detected, rules with regard to the pattern can be generated straight away. In addition, CMAR makes use of the class label distribution to prune. Given a frequent pattern P, let us assume c is the most dominant/mojority class in the set of examples matching P. If the number of examples having class label c and matching P, is less than the support threshold, then there is no need to search any superpattern (superset) P' of P. This is very clear as any rule in the form of $P' \to c$ cannot satisfy the support threshold either as superset P' will have no larger support than pattern P.

Once rules are mined from given the transaction data, CMAR builds a CR-tree to save space in storing rules as well as to search for rules efficiently. CMAR also performs a rule pruning step to remove redundant and noise rules. In particular, three principles were used for rule pruning, including 1) use more general and high-confidence rules to prune those more specific and lower confidence ones; 2) select only positively correlated rules based on χ^2 testing; 3) prune rules based on database coverage.

Finally, in the *classification* phase, for a given test example, CMAR extracts a subset of rules matching the test example and predicts its class label by analyzing this subset of rules. CMAR first groups rules according to their class labels and then finds the *strongest* group to perform classification. It uses a weighted χ^2 measure [30] to integrate both information of intra-group rule correlation and popularity. In other words, if those rules in a group are highly positively correlated and have good support, then the group has higher *strength.*

2. XRules

Different from CBA and CMAR which are applied to transaction data sets consisting of multi-dimensional records, XRules [44] on the other hand builds a structural rule-based classifier for semi-structured data, e.g., XML. In the training stage, it constructs *structural rules* which indicate what kind of structural patterns in an XML document are closely related to a particular class label. In the testing stage, it employs these structural rules to perform the structural classification.

Based on the definition of structural rules, XRules performed the following three steps during the training stage. 1) Mine frequent structural rules specific to each class using its proposed XMiner (which extends TreeMiner to find all frequent trees related to some class), with sufficient support and strength. Note that users need to provide a minimum support π_i^{min} for each class c_i. 2) Prioritize or order the rules in decreasing level of precedence as well as remove unpredictive rules. 3) Determine a special class called *default-class* which will be used to classify those test examples when none of the mined structural rules is applicable. After training, the classification model consists of an ordered rule set, and a *default-class.*

Finally, the testing stage performs classification on the given test examples without class labels. Given a test example S, there are two main steps for its classification, includingi.e., the *rule retrieval* step which finds all matching rules (stored in set $R(S)$) for S, as well as *class prediction* step which combines the statistics from each matching rule in $R(S)$ to predict the most likely class for S. Particularly, if $R(S) = \emptyset$, i.e. there are no matching rules, then default class is assigned to S; otherwise, $R(S) \neq \emptyset$. Assume $R_i(S)$ represent the matching rules in $R(S)$ with class c_i as their consequents. XRules used an average confidence method, i.e. for each class c_i, it computes the average rule strength for all the rules in $R_i(S)$. If the average rule strength for class c_i is big enough, the algorithm assigns the class c_i to the test example S. If the average rule strengths for all the classes are all very small, then the *default class* is used again to assign to S.

1.4 Applications

In this section, we briefly introduce some applications of applying rule based classification methods in text categorization [51], intrusion detection [74], diagnostic data mining [25], as well as gene expression data mining [50]

1.4.1 Text Categorization

It is well-known that Support Vector Machines (SVM) [57], Naïve Bayesian (NB) [58], and Rocchio's algorithm [60] are among the most popular techniques for text categorization, also called text classification. Their variations have also been applied to different types of learning tasks, e.g., learning with positive and unlabeled examples (PU learning) [64] [59] [61], [62], [63]. However, these existing techniques are typically used as black-boxes. Rule based classification techniques, on the other hand, can explain their classification results based on rules, and thus have also drawn a lot of attention. RIPPER [13], sleeping-experts [56], and decision tree-based rule induction systems [52] [53] [54] [55], have all been employed for text categorization.

Features used in the standard classification methods (such as SVM, Naïve Bayesian (NB), and Rocchio) are usually the individual terms in the form of words or word stems. Given a single word w in a document d, w's influence on d's predicted class is assumed to be independent of other words in d [70].

This assumption does not hold since w's *context*, encoded by the other words present in the document d, typically can provide more specific meanings and better indications on the d's classification, than w itself. As such, rule based systems, such as RIPPER [13] and sleeping-experts [56], have exploited *context* information of the words for text categorization [51]. Both techniques performed very well across different data sets, such as AP title corpus, TREC-AP corpus, and Reuters etc, outperforming classification methods, like decision tree [4] and Rocchio algorithm [60].

Next, we will introduce how RIPPER and sleeping-experts (or specifically sleeping-experts for phrases) make use of context information for text categorization, respectively.

RIPPER for text categorization

In RIPPER, the context of a word w_1 is a conjunction of the form

$$w_1 \in d \text{ and } w_2 \in d \ \ldots \text{ and } w_k \in d$$

Note that the context of a word w_1 consists of a number of other words w_2, ..., and w_k, that need to co-occur with w_1, but they may occur in any order, and in any location in document d.

Standard RIPPER algorithm was extended in the following two ways so that it can be better used for text categorization.

1. Allow users to specify a *loss ratio* [65]. A loss ratio is defined as the ratio of the cost

of a false negative to the cost of a false positive. The objective of the learning is to minimize misclassification cost on the unseen or test data. RIPPER can balance the recall and precision for a given class by setting a suitable loss ratio. Specifically, during the RIPPER's pruning and optimization stages, suitable weights are assigned to false positive errors and false negative errors respectively.

2. In text classification, while a large corpus or a document collection contains many different words, a particular document will usually only contain quite limited words. To save space for representation, a document is represented as a *single attribute* a, with its *value* as the set of words that appear in the document or a word list of the document, i.e. $a = \{w_1, w_2, ..., w_n\}$. The primitive tests (conditions) on a set-valued attribute a are in the form of $w_i \in a$.

For a rule construction, RIPPER will repeatedly adding conditions to rule r_0 which is initialized as an empty antecedent. Specifically, at each iteration i, a single condition is added to the rule r_i, producing an expanded rule r_{i+1}. The condition added to r_{i+1} is the one that maximizes information gain with regards to r_i. Given the set-valued attributes, RIPPER will carry out the following two steps to find a best condition to add:

1. For the current rule r_i, RIPPER will iterate over the set of examples/documents S that are covered by r_i and record a word list W where each word $w_i \in W$ appears as an element/value of attribute a in S. For each $w_i \in W$, RIPPER also computes two statistics, namely p_i and n_i, which represent the number of positive and negative examples in S that contain w_i respectively.

2. RIPPER will go over all the words $w_i \in W$, and use p_i and n_i to calculate the information gain for its condition $w_i \in a$. We can then choose the condition which yields the largest information gain and added it to r_i to form rule r_{i+1}.

The above process of adding new literals/conditions continues until the rule does not cover negative examples or until no condition has a positive information gain. Note the process only requires time linear in the size of S, facilitating its applications to handle large text corpora.

RIPPER has been used to classify or filter personal e-mails [69] based on a relatively small sets of labeled messages.

Sleeping-experts for phrases for text categorization

Sleeping-experts [56] is an ensemble framework which builds a *master algorithm* to integrate the "advice" of different "experts" or classifiers [76] [51]. Given a test example, the master algorithm uses a weighted combination of the predictions of the experts. One efficient weighted assignment algorithm is the *multiplicative* update method where weights for each individual experts are updated by multiplying them by a constant. Particularly, those "correct" experts that make right classification will be able to keep their weights unchanged (i.e. multiplying 1) while those "bad" experts that make wrong classification have to multiply a constant (less than 1) so that their weights will become smaller.

In the context of text classification, the experts correspond to all length-k phrases that occur in a corpus. Given a document that need to be classified, those experts are "awake" and make predictions if they appear in the document; the remaining experts are said to be "sleeping" on the document. Different from the context information used in the RIPPER, the context information in sleeping-experts (or sleeping-experts for phrases), is defined in the following phrase form

$$w_{i_1}, w_{i_2} \ldots w_{i_j}$$

where $i_1 < i_2 < \ldots < i_{j-1} < i_j$ and $i_j - i_1 < n$.

Note that there could be some "holes" or "gaps" between any two words in the context /phrase.

The detailed sleeping-experts for phrases algorithm is as follows.

The sleeping-experts algorithm for phrases

Input Parameters: $\beta \in (0,1), \theta_C \in (0,1)$, number of labeled documents T **Initialize**: $Pool \leftarrow \emptyset$

```
Do for $t$=1, 2, ..., $T$
    1. Receive a new document $w_1^t, w_2^t, \ldots, w_l^t$, and its classification $c^t$
    2. Define the set of active phrases:
        $W^t = \{\bar{w} | \bar{w} = w_{i_1}^t, w_{i_2}^t, \ldots, w_{i_j}^t,\ 1 \le i_1 < i_2 < \ldots < i_{j-1} < i_j < l,\ i_j - i_1 < n\}$
    3. Define the set of active mini-experts:
        $E^t = \{\bar{w}_k | \bar{w} \in W^t, k \in \{0,1\}\}$
    4. Initialize the weights of new mini-experts:
        $\forall \bar{w}_k \in E^t \ \ s.t. \ \ \bar{w}_k \notin Pool : \ \ p_{\bar{w}_k}^t = 1$
    5. Classify the document as positive if
        $y^t = \frac{\sum_{\bar{w} \in W^t} p_{\bar{w}_1}^t}{\sum_{\bar{w} \in W^t} \sum_{k=0,1} p_{\bar{w}_k}^t} > \theta_C$
    6. Update weights:
        $l(\bar{w}_k) = \begin{cases} 0, & \text{if } c^t = \text{k} \\ 1, & \text{if } c^t \neq k \end{cases} \quad \Rightarrow \quad p_{\bar{w}_k}^{t+1} = p_{\bar{w}_k}^t \times \beta^{l(\bar{w}_k)} = \begin{cases} p_{\bar{w}_k}^t, & \text{if } c^t = \text{k} \\ \beta \times p_{\bar{w}_k}^t, & \text{if } c^t \neq k \end{cases}$
    7. Renormalize weights:
        $(a) \ \ Z_t = \sum_{\bar{w}'_k \in E^t} p_{\bar{w}'_k}^t$
        $(b) \ \ Z_{t+1} = \sum_{\bar{w}'_k \in E^t} p_{\bar{w}'_k}^{t+1}$
        $(c) \ \ p_{\bar{w}'_k}^{t+1} = \frac{Z_t}{Z_{t+1}} p_{\bar{w}'_k}^{t+1}$
    8. Update: $Pool \leftarrow Pool \cup E^t$.
```

In this algorithm, the master algorithm maintains a *pool*, recording the sparse phrases appeared in the previous documents and a set **p**, containing one weight for each sparse phrase in the pool.

This algorithm iterates over all the T labeled examples to update the weight set **p**. Particularly, at each time step t, we have a document w_1^t, w_2^t, …, w_l^t with length l, and its classification label c^t (step 1). In step 2, we search for a set of active phrases, denoted by W^t from the given document. Step 3 defines two active mini-experts $\bar{w}_1$ and $\bar{w}_0$ for each phrase $\bar{w}$ where $\bar{w}_1$ ($\bar{w}_0$) predicts the document belongs to the class (does not belong to the class). Obviously, given the actual class label, only one of them is correct. In step 4, this algorithm initializes the weights of new mini-experts (not in the pool) as 1. Step 5 classifies the document by calculating the weighted sum of the min-experts and storing the sum into the variable y^t — the document is classified as positive (class 1) if $y^t > \theta_C$; otherwise the negative (class 0). $\theta_C = \frac{1}{2}$ has been set to minimize the errors and get a balanced precision and recall. After performing classification, Step 6 updates weights to reflect the correlation between the classification results and the actual class label. It first computes the *loss* $l(\bar{w}_k)$ of each mini-expert $\bar{w}_k$ — if the predicted label is equal to the actual label, then the loss $l(\bar{w}_k)$ is zero; 1 otherwise. The weight of each expert is then multiplied by a factor $\beta^{l(\bar{w}_k)}$

where $\beta < 1$ is called the learning rate, which controls how quickly the weights are updates. The value for β is in the range [0.1,0.5]. Basically, this algorithm keeps the weight of the correctly classified mini-expert unchanged but lower the weight of the wrongly classified mini-expert by multiplying β. Finally, step 7 normalizes the active mini-experts so that the total weight of the active mini-experts does not change. In effect, this re-normalization is to increase the weights of the mini-experts that were correct in classification.

1.4.2 Intrusion Detection

Nowadays, network-based computer systems play crucial roles in the society. However, criminals have attempted to intrude into and compromise these systems in various manners. According to Heady [72], an intrusion is defined as any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource, e.g. illegally accessing administrator or superuser privilege, attacking and rendering a system out of services etc. While some intrusion prevention techniques, such as user authentication by using passwords or biometrics as well as information protection by encryption, have been applied, they are not sufficient enough to address this problem as these systems typically have weaknesses due to their designs and programming errors [73]. As such, intrusion detection systems are thus imperative to serve as an additional shield to protect these computer systems from malicious activities or policy violations by closely monitoring the network and system activities.

There are some existing intrusion detection systems, which are manually constructed to protect a computer system based on some prior knowledge, such as known intrusion behaviors and the current computer system information. However, when facing new computer system environments and newly designed attacking/intruding methods, these types of *manual* and *ad hoc* intrusion detection systems are not flexible enough and will not be effective any more due to their limited adaptability.

We introduce a generic framework for building an intrusion detection system by analyzing the audit data [74], which refers to time-stamped data streams that can be used for detecting intrusions. The system first mines the audit data to detect the frequent activity patterns which are in turn used to guide the selection of system features as well as construction of additional temporal and statistical features. Classifiers can then be built based on these features and served as intrusion detection models to classify whether an observed system activity is legitimate or intrusive. Compared with those methods with hand crafted intrusion signatures to represent the intrusive activities, the approach has more generalized detection capabilities.

In general, there are two types of intrusion detection techniques, namely, anomaly detection and misuse detection. Anomaly detection determines whether deviation from an established normal behavior profile is an intrusion. In particular, a profile typically comprises of a few statistical measures on system activities, e.g. frequency of system commands during a user login session and CPU usage. Deviation from a profile can then be calculated as the weighted sum of the deviations of the constituent statistical measures. Essentially, this is an unsupervised method as it does not need users to provide known specific intrusions to learn from, and it can detect unknown, abnormal, and suspicious activities. The challenging issue for anomaly detection is how to define and maintain normal profiles — improper definition, such as lack of sufficient examples to represent different types of normal activities, could lead to high level false alarms, i.e. some non-intrusion activities are flagged as intrusions.

Misuse detection, on the other hand, exploits known intrusion activities/patterns (e.g. more than 3 consecutive failed logins within a few minutes is a penetration attempt) or weak spots of a system (e.g. system utilities that have the "buffer overflow" vulnerabilities) as training data to identify intrusions. Compared with anomaly detection, misuse detection is a supervised learning method, which can be used to identify those known intrusions effectively and efficiently as long as they are similar to the training intrusions. However, it can not detect unknown or newly invented attacks which could lead to unacceptable false negative error rates, i.e. some real intrusions are not able to be detected.

In order to perform intrusion predictions, we need to access those rich audit data which record system activities/events, the evidence of legitimate and intrusive user, as well as program activities. Anomaly detection searches for the normal usage patterns from the audit data while misuse detection encodes and matches intrusion patterns using the audit data [74].

For example, anomaly detection was performed for system programs [74], such as *sendmail*, as intruders use them to perform additional malicious activities. From the sequence of run-time system calls (e.g. open, read, etc), the audit data were segmented into a list of records and each of which has 11 consecutive system calls. RIPPER has been employed to detect rules which serve as normal (execution) profile. In total, 252 rules are mined to characterize the normal co-occurrences of these system calls and to identify the intrusions that deviate from the normal system calls.

In addition, another type of intrusions, where intruders aim to disrupt network services by attacking the weakness in TCP/IP protocols, has also been identified [74]. By processing the raw packet-level data into a time series of connection-level records which capture the connection information such as duration, number of bytes transferred in each direction, and the flag which specifies whether there is an error according to the protocol etc. Once again, RIPPER has been applied to mine 20 rules which serve as normal network profile, characterizing the normal traffic patterns for each network service. Given that the temporal nature of activity sequences [75], the temporal measures over features and the sequential correlation of features are particularly useful for accurate identification. Note the above anomaly detection methods need sufficient data which can cover as much variation of the normal behaviors as possible. Otherwise, given insufficient audit data, the anomaly detection will not be successful as some normal activities will be flagged as intrusions.

1.4.3 Using Class Association Rules for Diagnostic Data Mining

Liu et al [25] reported a deployed data mining system for Motorola, called Opportunity Map, that is based on class association rules mined from CBA [30]. The original objective of the system was to identify causes of cellular phone call failures. Since its deployment in 2006, it has been used for all kinds of applications.

The original data set contained cellular phone call records, and has more than 600 attributes and millions of records. After some pre-processing by domain experts, about 200 attributes are regarded as relevant to call failures. The data set is like any classification data set. Some of the attributes are continuous and some are discrete. One attribute indicates the final disposition of the call such as failed during setup, dropped while in progress, and ended successfully. This attribute is the class attribute in classification with discrete values. Two types of mining are usually performed with this kind of data:

1. Predictive data mining: The objective is to build predictive or classification models that can be used to classify future cases or to predict the classes of future cases. This has been the focus of research of the machine learning community.

2. Diagnostic data mining: The objective here is usually to understand the data and to find causes of some problems in order to solve the problems. No prediction or classification is needed.

In the above example, the problems are failed during setup and dropped while in progress. A large number of data mining applications in engineering domains are of this type because product improvement is the key task. The above application falls into the second type. The objective is not prediction, but to better understand the data and to find causes of call failures or to identify situations in which calls are more likely to fail. That is, the user wants interesting and actionable knowledge. Clearly, the discovered knowledge has to be understandable. Class association rules are suitable for this application.

It is easy to see that such kind of rules can be produced by classification algorithms such as decision trees and rule induction (e.g., CN2 and RIPPER), but they are not suitable for the task due to three main reasons:

1. A typical classification algorithm only finds a very small subset of the rules that exist in data. Most of the rules are not discovered because their objective is to find only enough rules for classification. However, the subset of discovered rules may not be useful in the application. Those useful rules are left undiscovered. We call this the completeness problem.

2. Due to the completeness problem, the context information of rules are lost, which makes rule analysis later very difficult as the user does not see the complete information.

3. Since the rules are for classification purposes, they usually contain many conditions in order to achieve high accuracy. Long rules are, however, of limited use according to our experience because the engineers can hardly take any action based on them. Furthermore, the data coverage of long rules is often so small that it is not worth doing anything about them.

Class association rule mining [30] is found to be more suitable as it generates all rules. The Opportunity Map system basically enables the user to visualize class association rules in all kinds of ways through OLAP operations in order to find those interesting rules that meet the user needs.

1.4.4 Gene Expression Data Analysis

In recent years, association rule mining techniques have been applied in bioinformatics domain, e.g. detecting patterns, clustering or classifying gene expression data [66] [39] [50]. Microarray technology enables us to measure the expression levels of tens of thousands of genes in cell simultaneously [66] and has been applied in various clinical research [39]. The gene expression datasets generated by microarray technology typically contain a large number of columns (corresponding to tens of thousands of human genes) but much smaller number of rows (corresponding to only tens or hundreds of conditions), which can be considered as tens or hundreds of very high-dimensional data. This is in contrast to those typical transaction databases, which have much more rows (e.g. millions of transactions) than columns (tens or hundreds of features).

The objective of microarray dataset analysis is to detect important correlations between gene expression patterns (genes and their corresponding expression value ranges) and disease outcomes (certain cancer or normal status) which are very useful biomedical knowledge and can be utilized for clinical diagnostic purposes [67] [68].

The rules that can be detected from gene expression data are in the following form:

$$gene_1[a_1, b_1], \ldots, gene_n[a_n, b_n] \rightarrow class \tag{1.11}$$

where $gene_i$ is the name of a gene and $[a_i, b_i]$ is its expression value range or interval. In other words, the antecedent of the rule in equation 1.11 consists of a set of conjunctive

TABLE 1.3: Example of gene expression data and rule groups

Rows/conditions	*Discretized gene expression data*	*Class label*
r1	a, b, c, d, e	C
r2	a, b, c, o, p	C
r3	c, d, e, f, g	C
r4	c, d, e, f, g	¬C
r5	e, f, g, h, o	¬C

gene expression level intervals and the consequent is a single class label. For example, $X95735[-\infty, 994] \rightarrow ALL$ is a rule that was discovered from the gene expression profiles of ALL/AML tissues [50]. It has only 1 condition for gene $X95735$ whose expression value is less than 994. We have two classes for the dataset where class ALL stands for *Acute Lymphocytic Leukemia* cancer and AML stands for *Acute Myelogenous Leukemia* cancer. Obviously, association rules are very useful in analyzing gene expression data. The discovered rules, due to their simplicity, can be easily interpreted by clinicians and biologists, which provide direct insights and potential knowledge that could be used for medical diagnostic purpose. This is quite different from other machine learning methods, such as Support Vector Machines (SVM), which typically serve as a black box in many applications. Although they could be more accurate in certain datasets for classification purposes, it is almost impossible to convince clinicians to adopt their predictions for diagnostic in practice, as the logics behind the prediction are hard to explain compared with rule based methods.

RCBT [50], Refined Classification Based on Top-k covering rule *groups* (TopkRGS), was proposed to address two challenging issues in mining the gene expression data. Firstly, huge number of rules can be mined from the high-dimensional gene expression dataset, even with rather high minimum support and confidence thresholds. It will be extremely difficult for biologists/clinicians to dig out clinically useful rules or diagnostic knowledge from large amount of rules. Secondly, the high dimensionality (tens of thousands of genes) and the huge number of rules lead to extremely long mining process.

To address the above challenging problems, RCBT discovers the most significant Top-kRGS for *each row* of a gene expression dataset. Note that TopkRGS can provide a more complete description for each row, which is different from existing interestingness measures that may fail to discover any interesting rules to cover some of the rows if given a higher support threshold. As such, the information in those rows that are not covered will not be captured in the set of rules. Given that gene expression datasets have a small number of rows, RCBT will not lose important knowledge.

Particularly, the rule group conceptually clusters rules from the same set of rows. We use the example in Table 1.3 to illustrate the concept of a rule group [50]. Note the gene expression data in Table 1.3 have been discretized. It consists of 5 rows, namely, $r1$, $r2$, ..., $r5$ where the first three rows have class label C while the last two have label ¬C. Given a item set I, its Item Support Set, denoted $R(I)$, is defined as the largest set of rows that contain I. For example, given item set $I = \{a, b\}$, its Item Support Set, $R(I) = \{r1, r2\}$. In fact, we observe that $R(a) = R(b) = R(ab) = R(ac) = R(bc) = R(abc) = \{r1, r2\}$. As such, they make up a rule group $\{a \rightarrow C$, $b \rightarrow C$, ..., $abc \rightarrow C\}$ of consequent C, with the upper bound $abc \rightarrow C$ and the lower bounds $a \rightarrow C$, and $b \rightarrow C$.

Obviously all rules in the same rule group have the exactly same support and confidence since they are essentially derived from the same subset of rows [50], i.e. $\{r1, r2\}$ in the above example. We can easily identify the remaining rule members based on the upper bound and all the lower bounds of a rule group. In addition, the significance of different rule groups can be evaluated based on both their confidence and support scores.

In addition, RCBT has designed row enumeration technique as well as several pruning strategies which make the rule mining process very efficient. A classifier has been constructed from the top-k covering rule groups. Given a test instance, RCBT also aims to reduce the chance of classifying it based on the *default class* by building additional stand-by classifiers. Specifically, given k sets of rule groups $RG1, \ldots, RGk$, k classifiers $CL1, ..., CLk$ are built where $CL1$ is the main classifier and $CL2, \ldots, CLk$ are stand-by classifiers. It makes a final classification decision by aggregating voting scores from all the classifiers.

A number of experiments have been carried out on real bioinformatics datasets, showing that RCBT algorithm is orders of magnitude faster than previous association rule mining algorithms.

1.5 Discussion and Conclusion

In this chapter, we discussed two types of popular rule-based classification approaches, i.e., rule induction and classification based on association rules. Rule induction algorithms generate a small set of rules directly from the data. Well-known systems include AQ by Michalski et al. [36], CN2 by Clark and Niblett [9], FOIL by Quinlan [10], FOCL by Pazzani et al. [37], I-REP by Furnkranz and Widmer [11], and RIPPER by Cohen [13]. Using association rules to build classifiers was proposed by Liu et al. in [30], which also reported the CBA system. CBA selects a small subset of class association rules as the classifier. Other classifier building techniques include combining multiple rules by Li et al. [31], using rules as features by Meretakis and Wthrich [38], Antonie and Zaiane [32], Deshpande and Karypis [33], and Lesh et al. [35], generating a subset of rules by Cong et al. [39], Wang et al. [40], Yin and Han [41], and Zaki and Aggarwal [44]. Additional systems include those by Li et al. [45], Yang et al. [46], etc.

Note well-known decision tree methods [4], such as ID3 and C4.5, build a tree structure for classification. The tree has two different types of nodes, namely decision nodes (internal nodes) and leaf nodes. A decision node specifies a test based on a single attribute while a leaf node indicates a class label. A decision tree can also be converted to a set of IF-THEN rules. Specifically, each path from the root to a leaf forms a rule where all the decision nodes along the path form the conditions of the rule and the leaf node forms the consequent of the rule. The main differences between decision tree and rule induction are in their *learning strategy* and *rule understandability.* Decision tree learning uses the divide-and-conquer strategy. In particular, at each step, all attributes are evaluated and one is selected to partition/divide the data into m disjoint subsets, where m is the number of values of the attribute. Rule induction, however, uses the separate-and-conquer strategy, which evaluates all attribute-value pairs (conditions) and selects only one. Thus, each step of divide-and-conquer expands m rules, while each step of separate-and-conquer expands only one rule. On top of that, the number of attribute-value pairs are much larger than the number of attributes. Due to these two effects, the separate-and-conquer strategy is much slower than the divide-and-conquer strategy. In terms of rule understandability, while if-then rules are easy to understand by human beings, we should be cautious about rules generated by rule induction (e.g. using the sequential covering strategy) since they are generated in order. Such rules can be misleading because the covered data are removed after each rule is generated. Thus the rules in the rule list are not independent of each other. In addition, A rule r may be of high quality in the context of the data D' from which r was generated. However, it may be a very weak rule with a very low accuracy (confidence) in the context of the whole data set D ($D' \subseteq D$) because many training examples that can be covered by r have already been removed by rules generated before r. If you want to understand the rules generated by rule induction and possibly use them in some real-world applications, you should be aware of this fact. The rules from decision trees, on the other hand, are independent of each other and are also mutually exclusive. The main differences between decision tree (or a rule induction system) and class association rules (CARs) are in their mining algorithms and the final rule sets. CARs mining detects all rules in data that satisfy the user-specified minimum support (minsup) and minimum confidence (minconf) constraints while a decision tree or a rule induction system detects only a small subset of the rules for classification. In many real-world applications, rules that are not found in the decision tree (or a rule list) may be able to perform classification more accurately. Empirical comparisons have demonstrated that in many cases, classification based on CARs performs more accurately than decision trees and rule induction systems.

The complete set of rules from CARs mining could also be beneficial from a rule usage point of view. For example, in a real-world application for finding causes of product problems (e.g. for diagnostic purposes), more rules are preferred to fewer rules because with more rules, the user is more likely to find rules that indicate the causes of the problems. Such rules may not be generated by a decision tree or a rule induction system. A deployed data mining system based on CARs is reported in [25]. Finally, CARs mining, like standard association rule mining, can only take discrete attributes for its rule mining, while decision trees can deal with continuous attributes naturally. Similarly, rule induction can also use continuous attributes. But for CARs mining, we first need to apply an attribute discretization algorithm to automatically discretize the value range of a continuous attribute into suitable intervals [47], [48], which are then considered as discrete values to be used for CARs mining algorithms. This is not a problem as there are many discretization algorithms avalable.

Bibliography

[1] Li X. L., Liu B., Ng, S.K. Learning to identify unexpected instances in the test set. In *Proceedings of Twentieth International Joint Conference on Artificial Intelligence*, pages 2802–2807, India, 2007.

[2] Cortes Corinna, and Vapnik, Vladimir N. Support-Vector Networks. *Machine learning*, 20 (3):273–297, 1995.

[3] Hopfield J. J. Neural networks and physical systems with emergent collective computational abilities. NatL Acad. Sci. USA, 79 (8): 2554–2558, 1982.

[4] Quinlan, J. C4. 5: programs for machine learning. Morgan Kaufmann Publishers, 1993.

[5] George H. John and Pat L. Estimating Continuous Distributions in Bayesian Classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, San Mateo, 1995.

[6] Bremner D., Demaine E., Erickson J., Iacono J., Langerman S., Morin P., and Toussaint G. Output-sensitive algorithms for computing nearest-neighbor decision boundaries. *Discrete and Computational Geometry*, 33 (4): 593-604, 2005.

[7] Hosmer, David W., Lemeshow, and Stanley. Applied Logistic Regression. Wiley, 2000.

[8] Rivest, R. Learning decision lists. *Machine learning*, 2(3): 229–246, 1987.

[9] Clark, P. and Niblett T. The CN2 induction algorithm. *Machine learning*, 3(4): 261–283, 1989.

[10] Quinlan, J. Learning logical definitions from relations. *Machine learning*, 5(3): 239-266, 1990.

[11] Frnkranz, J. and Widmer G. Incremental reduced error pruning. In *Proceedings of International Conference on Machine Learning (ICML-1994)*, pages 70–77, 1994.

[12] Brunk, C. and Pazzani M. An investigation of noise-tolerant relational concept learning algorithms. In *Proceedings of International Workshop on Macine Learning*, pages 389–393, 1991.

[13] Cohen W. W. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[14] Mitchell, T. Machine Learning. McGraw Hill. 1997.

[15] Donald E. K. The Art of Computer Programming. Addison-Wesley, 1968.

[16] Agrawal, R., Imieliski T., and Swami A. Mining association rules between sets of items in large databases. In Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD-1993), pages 207–216, 1993.

[17] Agrawal, R. and Srikant R. Fast algorithms for mining association rules in large databases. In *Proceedings of International Conference on Very Large Data Bases (VLDB-1994)*, pages 487–499, 1994.

[18] Michalski, R. S. On the quasi-minimal solution of the general covering problem. In *Proceddings of the Fifth International Symposium on Information Processing*, pages 125–128, 1969.

[19] Han J. W., Kamber M., and Pei J. *Data Mining: Concepts and Technqiues.* 3rd edition, Morgan Kaufmann, 20011.

[20] Liu B. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data.* Springer, 2006.

[21] Han J. W., Pei J., and Yin Y. *Mining frequent patterns without candidate generation.* In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD-2000)*, pages 1–12, 2000.

[22] Bayardo Jr, R. and Agrawal R. *Mining the most interesting rules.* In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-1999)*, pages 145–154, 1999.

[23] Klemettinen, M., Mannila H., Ronkainen P., Toivonen H., and Verkamo A. *Finding interesting rules from large sets of discovered association rules.* In *Proceedings of ACM International Conference on Information and Knowledge Management (CIKM-1994)*, pages 401–407, 1994.

[24] Liu B., Hsu W., and Ma Y. *Pruning and summarizing the discovered associations.* In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-1999)*, pages 125–134, 1999.

[25] Liu B., Zhao K., Benkler J., and Xiao W. *Rule interestingness analysis using OLAP operations.* In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2006)*, pages 297–306, 2006.

[26] Padmanabhan, B. and Tuzhilin A. *Small is beautiful: discovering the minimal set of unexpected patterns.* In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2000)*, pages 54–63, 2000.

[27] Piatetsky-Shapiro, G. *Discovery, analysis, and presentation of strong rules. Knowledge discovery in databases*, pages 229–248, 1991.

[28] Silberschatz, A. and Tuzhilin A. *What makes patterns interesting in knowledge discovery systems. IEEE Transactions on Knowledge and Data Engineering*, 8 (6): 970–974, 1996.

[29] Tan P., Kumar V., and Srivastava J. *Selecting the right interestingness measure for association patterns.* In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, pages 32-41, 2002.

[30] Liu B., Hsu W., and Ma Y. *Integrating classification and association rule mining.* In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-1998)*, pages 80–86, 1998.

[31] Li W., Han J., and Pei J. *CMAR: Accurate and efficient classification based on multiple class-association rules.* In *Proceedings of IEEE International Conference on Data Mining (ICDM-2001)*, pages 369–376, 2001.

[32] Antonie, M. and Zaïane O. *Text document categorization by term association.* In *Proceedings of IEEE International Conference on Data Minig (ICDM-2002)*, Pages, 19–26, 2002.

[33] Deshpande, M. and Karypis G. *Using conjunction of attribute values for classification.* In *Proceedings of ACM Intl. Conf. on Information and Knowledge Management (CIKM-2002)*, pages 356–364, 2002.

[34] Jindal, N. and Liu B. *Identifying comparative sentences in text documents.* In *Proceedings of ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR-2006)*, pages 244–251, 2006.

[35] Lesh, N., Zaki M. , and Ogihara M. *Mining features for sequence classification.* In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-1999)*, pages 342-346, 1999.

[36] Michalski, R., Mozetic I., Hong J., and Lavrac N. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of National Conf. on Artificial Intelligence (AAAI-86)*, pages 1041–1045, 1986.

[37] Pazzani, M., Brunk C., and Silverstein G. A knowledge-intensive approach to learning relational concepts. In *Proceedings of Intl. Workshop on Machine Learning (ML-1991)*, pages 432–436, 1991.

[38] Meretakis, D. and Wuthrich B. Extending naïve Bayes classifiers using long itemsets. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-1999)*, pages 165–174, 1999.

[39] Cong G., Tung AKH, X. Xu, Pan F., and Yang J. Farmer: Finding interesting rule groups in microarray datasets. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD-2004)*, pages 143–154, 2004.

[40] Wang, K., Zhou S., and He Y. Growing decision trees on support-less association rules. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2000)*, pages 265-269, 2000.

[41] Yin X. and Han J. CPAR: Classification based on predictive association rules. In *Proceedings of SIAM International Conference on Data Mining (SDM-2003)*, pages 331-335, 2003.

[42] Lin W., Alvarez S., and Ruiz C. Efficient adaptive-support association rule mining for recommender systems. *Data mining and knowledge discovery*, 6(1): 83-105, 2002.

[43] Mobasher, B., Dai H., Luo T., and Nakagawa M. Effective personalization based on association rule discovery from web usage data. In *Proceedings of ACM Workshop on Web Information and Data Management*, pages 9–15, 2001.

[44] Zaki, M. and Aggarwal C. XRules: an effective structural classifier for XML data. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)*, pages 316–325, 2003.

[45] Li J., Dong G., Ramamohanarao K., and Wong L. DeEPs: A new instance-based lazy discovery and classification system. *Machine learning*, 54(2): p. 99-124, 2004.

[46] Yang Q., Li T., and Wang K. Building association-rule based sequential classifiers for web-document prediction. *Data mining and knowledge discovery*, 8(3): 253–273, 2004.

[47] Dougherty, J., Kohavi R., and Sahami M. Supervised and unsupervised discretization of continuous features. In *Proceedings of International Conference on Machine Learning (ICML-1995)*, 194–202, 1995.

[48] Fayyad, U. and Irani K. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Intl. Joint Conf. on Artificial Intelligence (IJCAI-1993)*, pages 1022–1028, 1993.

[49] Zheng Z., Kohavi R., and Mason L. Real world performance of association rule algorithms. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001)*, pages 401-406, 2001.

[50] Cong G., Tan K.-L., Tung AKH, and Xu X. Mining top-k covering rule groups for gene expression data. In *Proceeding of the 2005 ACM-SIGMOD international conference on management of data (SIGMOD–05)*, pages 670-681, 2005.

[51] William W. Cohen and. Yoram S. Context-sensitive learning methods for text categorization. *ACM Transactions on Information Systems*, 17(2): 141–173, 1999.

[52] Johnson D. , Oles F. , Zhang T., and Goetz T. A Decision Tree-based Symbolic Rule Induction System for Text Categorization. *IBM Systems Journal*, 41(3), pp. 428–437, 2002.

[53] Apte C. , Damerau F. , and Weiss S. Automated Learning of Decision Rules for Text Categorization. *ACM Transactions on Information Systems*, 12(3), pp. 233–251, 1994.

[54] Weiss S. M., Apte C. , Damerau F., Johnson D., Oles F., Goetz T., and Hampp T. Maximizing text-mining performance. *IEEE Intelligent Systems*, 14(4), pp. 63–69, 1999.

[55] Weiss S. M., and Indurkhya N. Optimized Rule Induction. *IEEE Exp.*, 8(6), pp. 61–69, 1993.

[56] Freund, Y., Schapire, R., Singer, Y., and Warmuth, M. Using and combining predictors that specialize. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 334-343, 1997.

[57] Joachims T. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Proceedings of the European Conference on Machine Learning (ECML)*, pages 137–142, 1998.

[58] Andrew M., and Nigam K. A comparison of event models for Naïve Bayes text classification. In *Proceedings of AAAI-98 workshop on learning for text categorization.* Vol. 752. 1998.

[59] Liu B., Lee W. S., Yu P. S. and Li X. L. Partially Supervised Classification of Text Documents. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML-2002)*, pages 387–394, Australia, 2002.

[60] Rocchio. J. Relevant feedback in information retrieval. In *G. Salton (ed.). The smart retrieval system: experiments in automatic document processing*, pages 313–323, Englewood Cliffs, NJ, 1971.

[61] Li X. L. and Liu B. Learning to Classify Texts Using Positive and Unlabeled Data. In *Proceedings of Eighteenth International Joint Conference on Artificial Intelligence*, pages 587–592, Mexico, 1993.

[62] Li X. L., Liu B., Yu P. S., and Ng S. K. Positive unlabeled learning for data stream classification. In *Proceedings of the Ninth SIAM International Conference on Data Mining*, pages 257-268, USA, 2009.

[63] Li X. L., Liu B., Yu P. S., and Ng S. K. Negative Training Data Can Be Harmful to Text Classification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 218–228, USA, 2010.

[64] Liu B., Dai Y., Li X. L., Lee W. S., and Yu P. S. Building text classifiers using positive and unlabeled examples. In *Proceedings of Third IEEE International Conference on Data Mining*, pages 179-186, USA, 2003.

[65] Lewis, D. and Catlett, J. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the Eleventh Annual Conference on Machine Learning*, pages 148–156, USA, 1994.

[66] Li X. L., Tan Y. C., and Ng S. K. Systematic Gene Function Prediction from Gene Expression Data by Using a Fuzzy Nearest-Cluster Method *BMC Bioinformatics*, 7(Suppl 4): S23, 2006.

[67] Han X. X., and Li X. L. Multi-resolution Independent Component Analysis for High-Performance Tumor Classification and Biomarker Discovery *BMC Bioinformatics*, 12(Suppl 1):S7, 2011

[68] Yang P., Li X. L., Mei J. P., Kwoh C. K. and Ng S. K. Positive-Unlabeled Learning for Disease Gene Identification *Bioinformatics*, Vol 28(20), pages 2640–2647, 2012

[69] Cohen W. W. Learning Rules that Classify E-Mail. In *Proceedings of the AAAI Spring Symposium on Machine Learning in Information Access*, pages 18–25, 1996.

[70] Liu B., Dai Y., Li X. L., Lee W. S., and Yu P. S. Text classification by labeling words. In *Proceedings of the National Conference on Artificial Intelligence*, pages 425-430, USA, 2004.

[71] Cohen W. W. Learning Trees and Rules with Set-valued Features In *Proceedings of the thirteenth national conference on Artificial intelligence*, pages 709–716, 1996.

[72] Heady, R., G. Luger, A. Maccabe, and M. Servilla: The Architecture of a Network Level Intrusion Detection System. Technical report, University of New Mexico, 1990.

[73] Lee W., Stolfo S. J., and Mok K. W. Adaptive Intrusion Detection: A Data Mining Approach. *Artificial Intelligence Review - Issues on the application of data mining archive*, 14(6): 533 – 567, 2000.

[74] Lee, W. and Stolfo S. J. Data Mining Approaches for Intrusion Detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, pages 6–6, 1998.

[75] Mannila, H. and Toivonen H. Discovering Generalized Episodes Using Minimal Occurrences. In *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining.* Portland, Oregon, pages 146–151,1996.

[76] Friedman J.H., and Popescu B. E. Predictive learning via rule ensembles *The Annals of Applied Statistics*, 2(3): 916 – 954, 2008.