An Associative Classifier For Uncertain Datasets

Metanat Hooshsadat and Osmar R. Zaïane

University of Alberta
Edmonton, Alberta, Canada
{hooshsad,zaiane}@cs.ualberta.ca

Abstract. The classification of uncertain datasets is an emerging research problem that has recently attracted significant attention. Some attempts to devise a classification model with uncertain training data have been proposed using decision trees, neural networks, or other approaches. Among those, the associative classifiers have inspired some of the uncertain classification algorithms given their promising results on standard datasets. We propose a novel associative classifier for uncertain data. Our method, Uncertain Associative Classifier (UAC) is efficient and has an effective rule pruning strategy. Our experimental results on real datasets show that in most cases, UAC reaches better accuracies than the state of the art algorithms.

1 Introduction

Typical relational databases or databases in general hold collections of records representing facts. These facts are observations with known values stored in the fields of each tuple of the database. In other words, the observation represented by a record is assumed to have taken place and the attribute values are assumed to be true. We call these databases "certain database" because we are certain about the recorded data and their values. In contrast to "certain" data there is also "uncertain data"; data for which we may not be sure about the observation whether it really took place or not, or data for which the attribute values are not ascertained with 100% probability.

Querying such data, particularly computing aggregations, ranking or discovering patterns in probabilistic data is a challenging feat. Many researchers have focused on uncertain databases, also called probabilistic databases, for managing uncertain data [1], top-k ranking uncertain data [2], querying uncertain data [3], or mining uncertain data [4, 5]. While many approches use an existancial uncertainty attached to a record as a whole, our model targets uncertain databases with probabilities attached to each attribute value.

This paper addresses the problem of devising an accurate rule-based classifier on uncertain training data. There are many classification paradigms but the classifiers of interest to our study are rule-based. We opted for associative classifiers, classifiers using a model based on association rules, as they were shown to be highly accurate and competitive with other approaches [6].

After briefly reviewing related work for associative classification as well as published work on classifying in the presence of uncertainty, we present in Section

3 our novel classification method UAC. Finally in Section 4 we present empirical evaluations comparing UAC with other published works.

2 Related Works

Recently, a considerable amount of studies in machine learning are directed toward the uncertain data classification, including: TSVC [7] (inspired by SVM), DTU [8] (decision tree), UNN [9] (based on Neural Network), a Bayesian classifier [10], uRule [11] (rule based), uHARMONY [12] and UCBA [13] (based on associative classifiers). However, models suggested by the previous work do not capture some possible types of uncertainty. In previous studies, numerical attributes are only modeled by intervals, while they may exist in other forms such as probability vectors. Categorical attributes are modeled by a probability distribution vector over their domain where the vector is unrealistically assumed to be completely known. We use a probability on each attribute value.

High accuracy and strong flexibility are some of the advantageous characteristics of the rule based classifiers. Investigating rule based uncertain data classifiers has been the theme of many studies. One of these studies is uRule [11], which defines the information gain metric in presence of uncertainty. The probability of each rule classifying the instance is computed based on the weighting system introduced by uRule.

Associative classification is a large category of rule based classification in which the rule induction procedure is based on the association rule mining technique. Some of the prominent associative classifiers are CBA [14], ARC [15], and CMAR [16]. In this paper, we introduce an associative classifier for uncertain datasets, which is based on CBA. CBA is highly accurate, flexible and efficient both in time and memory [14].

CBA directly adopts Apriori to mine the potential classification rules or strong *ruleitems* from the data. Ruleitems are those association rules of form $a \rightarrow c$, where the consequence (c) is a class label and the antecedent (a) is a set of *attribute assignments*. Each *attribute assignment* consists of an attribute and a value which belongs to the domain of that attribute. For example, if A_1 and A_2 are two attributes and c is a class label, $r = (A_1 : u_1, A_2 : u_2 \rightarrow c)$ is a ruleitem. r implies that if A_1 and A_2 have values of u_1 and u_2 respectively, the class label should be c. A ruleitem is strong if its support and confidence are above the predefined thresholds.

After mining the strong ruleitems, a large number of them are eliminated by applying the *database coverage* approach. This method of filtering rules is applied by all rule-based classifiers, particularly associative classifiers. However, in the case of uncertain data, database coverage presents a significant challenge. Rule based classifiers often need to evaluate various rules to pick the best ones. This level is critical in maintaining a high accuracy. The evaluation often involves the answer to the following question: *To which training instances can a rule be applied?* Yet, the answer is not obvious for uncertain datasets. Many uncertain dataset instances may satisfy the antecedent of a rule, each with a different

probability. Existing uncertain data rule based classifiers have suggested various answers to this problem.

uHARMONY suggested a lower bound on the probability by which the instance satisfies the rule antecedent. This approach is simple and fast, but the difficulty or even impossibility of setting the threshold is a problem. This is explained in more detail in Section 3.2. uRule suggested to remove the items in the antecedent of the rule from the instance, to leave only the uncovered part of the instance every time. In contrast to uHARMONY, this method uses the whole dataset but it may cause sensitivity to noise which is undesirable. UCBA, wich is based on CBA, does not include the uncertainty in the rule selection process; they select as many rules as possible. This method does not filter enough rules; so may decrease the accuracy.

In UAC, we introduce a new solution to the coverage problem. This computation does not increase the running time complexity and needs no extra passes over the dataset.

3 UAC Algorithm

In this section, we present our novel algorithm, *UAC*. Before applying UAC to uncertain numerical attributes in the train sets, they are first transformed into uncertain categorical attributes using U-CAIM [10], assuming the normal distribution on the intervals. After discretization, the value of the i-th attribute for the j-th instance is a list of value-probability pairs, as shown in Equation 1.

$$\begin{array}{l} A_{j,i} = \{(x_{j,i,1} : p_{j,i,1}), (x_{j,i,2} : p_{j,i,2}), .., (x_{j,i,k} : p_{j,i,k})\} \\ \forall q \leq k \, ; A_j.l \leq x_{j,i,q} \leq A_j.u \; \Sigma_{q=1}^{k} p_{j,i,q} = 1. \end{array} \tag{1}$$

Building an associative classifier consists of two distinct steps: 1- Rule Extraction, 2- Rule Filtering. In this section each step of UAC is explained. Later, the procedure of classifying a new test instance is described.

3.1 Rule Extraction

In uncertain datasets, an association rule is considered strong if it is frequent and its *confidence (Conf)* is above a user defined threshold called *minimum confidence*. A ruleitem is frequent if its *Expected Support (ES)* is above a user defined threshold called *minimum expected support*. The definitions of the expected support and the confidence are as follows.

Definition If a is an itemset and c is a class label, expected support (ES) and confidence (Conf) of a ruleitem are calculated by Equation 2. Here, the ruleitem is denoted by $r = a \rightarrow c$ and T is the set of all transactions.

$$\begin{array}{ll} ES(a) & = \Sigma_{\forall t \in T} \Pi_{\forall i \in a} P(i \in t) \\ ES(a \rightarrow c) & = \Sigma_{\forall t \in T, t.class = c} \Pi_{\forall i \in a} P(i \in t). \\ Conf(a \rightarrow c) & = \frac{ES(a \rightarrow c)}{ES(a)}. \end{array} \tag{2}$$

Some studies have criticized expected support and defined another measure which is called probabilistic support [17] [18]. Probabilistic support is defined as the probability of an itemset to be frequent with respect to a certain minimum expected support. However, probabilistic support increases the time complexity significantly. Therefore to be more efficient, UAC uses the expected support.

uHARMONY defines another measure instead of confidence which is called *expected confidence*. The computation of this measure takes $O(|T|^2)$ time where $|T|$ is the number of instances. Computing confidence is only $O(1)$, thus we use confidence for efficiency reasons. Our experimental results in Section 4 empirically shows that our confidence based method can reach high accuracies.

Our rule extraction method is based on UApriori [4]. The candidate set is first initialized by all rules of form $a \rightarrow c$ where a is a single *attribute assignment* and c is a class label. After removing all infrequent ruleitems, the set of candidates is pruned by the pessimistic error rate method [19]. Each two frequent ruleitems with the same class label are then joined together to form the next level candidate set. The procedure is repeated until the generated candidate set is empty, meaning all the frequent ruleitems have been found. Those ruleitems that are strong (their confidence is above the predefined threshold) are the potential classification rules. In the next section, the potential ruleitems are filtered and the final set of rules is formed.

3.2 Rule Filtering

The outcome of the rule extraction is a set of rules called *rawSet*. Usually the number of ruleitems in rawSet is excessive. Excessive rules may have negative impact on the accuracy of the classification model. To prevent this, UAC uses the database coverage method to reduce the set of rules while handling the uncertainty. The initial step of the database coverage method in UAC is to sort rules based on their absolute precedence to accelerate the algorithm. Absolute precedence in the context of uncertain data is defined as follows:

Definition: Rule r_i has absolute precedence over rule r_j or $r_i \succ r_j$, if *a*) r_i has higher confidence than r_j; *b*) r_i and r_j have the same confidence but r_i has higher expected support than r_j; *c*) r_i and r_j have the same confidence and the same expected support but r_i have less items in its antecedent than r_j.

When data is not certain, confidence is a good and sufficient measure to examine whether a rule is the best classifier for an instance. But when uncertainty is present, there is an additional parameter in effect. To illustrate this issue, assume rules $r_1 : [m, t \rightarrow c_1]$ and $r_2 : [n \rightarrow c_2]$ having confidences of 0.8 and 0.7, respectively. It is evident that $r_1 \succ r_2$. However, for a test instance like $I_1 : [(m : 0.4), (n : 0.6), (t, 0.3) \rightarrow x]$ where x is to be predicted, which rule should be used? According to CBA, r_1 should be used because its confidence is higher than that of r_2. However, the probability that I_1 satisfies the antecedent of r_1 is small, so r_1 is not likely to be the right classifier. We solve this problem by including another measure called *PI*. *PI* or *probability of inclusion*, denoted by $\pi(r_i, I_k)$, is described as the probability by which rule r_i can classify instance I_k. *PI* is

defined in Equation 3. In the example above $\pi(r_1, I_1)$ is only $0.3 \times 0.4 = 0.12$, While $\pi(r_2, I_1)$ is 0.6.

$$\pi(r_i, I_k) = \Pi_{w \in r_i} P(w \in I_k). \tag{3}$$

Next, we define *applicability*, denoted by $\alpha(r_i, I_k)$ in Equation 4. Applicability is the probability by which rule r_i correctly classifies instance I_k and is used as one of the main metrics in UAC. For the previous example, $\alpha(r_1, I_1) = 0.096$ and $\alpha(r_2, I_1) = 0.42$. Thus, it is more probable that I_1 is correctly classified by r_2 than r_1.

$$\alpha(r_i, I_k) = r_i.Conf \times \pi(r_i, I_k). \tag{4}$$

Now based on the applicability, we define the concept of relative precedence of rule r_i over rule r_j with respect to I_k. This is denoted by $r_i \succ_{[I_k]} r_j$ and is defined as follows:

Definition: Rule r_i has relative precedence over rule r_j with respect to instance I_k denoted by $r_i \succ_{[I_k]} r_j$, if: *a)* $\alpha(r_i, I_k) > \alpha(r_j, I_k)$ *b)* r_i and r_j have the same applicability with respect to I_k but r_i has absolute precedence over r_j.

Having $r_i \succ_{[I_k]} r_j$ implies that r_i is "more reliable" than r_j in classifying I_k. It is evident from the definition, that the concept of "more reliable" rule in an uncertain data classifier is relative. One rule can be more reliable than the other when dealing with an instance, and the opposite may be true for another instance. In the previous example, r_2 has relative precedence over r_1, even though r_1 has absolute precedence over r_2.

UAC uses the relative precedence as well as the absolute precedence to filter rawSet. The database coverage algorithm of UAC has 3 stages that are explained below.

Stage 1: Finding ucRules and uwRules After sorting rawSet based on the absolute precedence, we make one pass over the dataset to link each instance i in the dataset to two rules in rawSet: *ucRule* and *uwRule*. *ucRule* is the rule with the highest relative precedence that correctly classifies i. In contrast, *uwRule* is the rule with the highest relative precedence that wrongly classifies i. The pseudocode for the first stage is presented in Algorithm 1.

In Algorithm 1, three sets are declared. U contains all the rules that classify at least one training instance correctly. Q is the set of all *ucRule*s which have relative precedence over their corresponding *uwRule*s with respect to the associated instances. If $i.uwRule$ has relative and absolute precedence over the corresponding *ucRule*, a record of form $< i.id, i.class, ucRule, uwRule >$ is put in A. Here, $i.id$ is the unique identifier of the instance and $i.class$ represents the class label.

To find the corresponding *ucRule* and *uwRule* for each instance, the procedure starts at the first rule of the sorted rawSet and descends. For example, if there is a rule that correctly classifies the target instance and has applicability of α, we pass this rule and look for the rules with higher applicabilities to assign as *ucRule*. Searching continues only until we reach a rule that has a confidence

of less than α. Clearly, this rule and rules after it (with less confidence) have no chance of being *ucRule*. The same applies to *uwRule*. Also as shown in Algorithm 1 lines 4 and 6, the applicability values of *ucRule* and *uwRule* are stored to expedite the process for the next stages.

The purpose of the database coverage in UAC is to find the best classifying rule (coverage) for each instance in the dataset. The covering rules are then contained in the final set of rules and others are filtered out. The best rule, that is the covering rule, in CBA is the highest precedence rule that classifies an instance. This definition is not sufficient for UAC because the highest precedence rule may have a small PI.

To solve the aforementioned problem, uHARMONY sets a predefined lower bound on the PI value of the covering rule, a method with various disadvantages. Clearly, not only estimating the suitable lower bound is critical, but it is also intricate, and even in many cases impossible. When predicting a label for an instance, rules that have higher PI than the lower bound are treated alike. To improve upon this, it is necessary to set the lower bound high enough to avoid low probability rules covering the instances. However, it remains that it is possible that the only classifying rules for some of the instances are not above that lower bound and are removed. Additionally, setting a predefined lower bound filters out usable information, while the purpose of the uncertain data classifiers is to use all of the available information. Moreover, having a single bound for all of the cases is not desirable. Different instances may need different lower bounds.

Given all the above reasons, we need to evaluate the suitable lower bound for each instance. The definition of the covering rule in UAC is as follows, where we use the applicability of $i.ucRule$ as our lower bound for covering i.

Definition: Rule r covers instance i if: *a*) r classifies at least one instance correctly; *b*) $\pi(r, i) > 0$; *c*) $\alpha(r, i) > \alpha(i.ucRule, i) = cApplic$. *d*) $r \succ i.ucRule$

cApplic represents the maximum rule applicability to classify an instance correctly. Thus, it is the suitable lower bound for the applicability of the covering rules. This will ensure that each instance is covered with the best classifying rule (*ucRule*) or a rule with higher relative and absolute precedence than *ucRule*. In the next two stages, we remove the rules that do not cover any instance from rawSet.

Stage 2: Managing Replacements In this stage (Algorithm 2), cases that were stored in A at Stage 1 are managed. A contains all cases where $i.uwRule$ has relative and absolute precedence over $i.ucRule$, thus $i.ucRule$ may not cover i. If $i.uwRule$ is flagged in Stage 1, i is covered by $i.uwRule$ (lines 3, 4, and 5). Otherwise based on the definition of the covering rule in Stage 1, i may get the coverage by the other rules such as w which have the following characteristics: *a*) w classifies i incorrectly; *b*) w has relative precedence over $i.ucRule$ with respect to i; *c*) w has absolute precedence over $i.ucRule$.

Function *allCoverRules* (line 7) finds all such rules as w within U, which are called the replacements of $i.ucRule$. The replacement relation is stored in a DAG (directed acyclic graph) called *RepDAG*. In RepDAG, each parent node

Algorithm 1 UAC Rule Filtering: Stage 1

```
1: Q = ∅; U = ∅; A = ∅
2: for all i ∈ Dataset do
3:     i.ucRule = firstCorrect(i)
4:     i.cApplic = α(i.ucRule, i)
5:     i.uwRule = firstWrong(i)
6:     i.wApplic = α(i.uwRule, i)
7:     U.add(ucRule)
8:     ucRule.covered[i.class] + +
9:     if (ucRule ≻[i] uwRule) and ucRule ≻ uwRule then
10:        Q.add(ucRule)
11:        flag(ucRule)
12:    else
13:        A.add(< i.id, i.class, ucRule, uwRule >)
14:    end if
15: end for
```

has a pointer to each child node via the *replace* set (line 12). The number of incoming edges is stored in *incom* (line 14). Each node represents a rule and each edge represents a replacement relation.

Each rule has a *covered* array in UAC where $r.covered[c]$ is used to store the total number of instances covered by r and labeled by class c. If $r.covered[r.class] = 0$, then r does not classify any training instance correctly and is filtered out. Starting from line 22, we traverse RepDAG in its topologically sorted order to update the *covered* array of each rule. Rule r_i comes before r_j in the sorted order, if $r_i \succ r_j$ and there is no instance such as I_k where $r_j \succ_{[I_k]} r_i$. If a rule fails to cover any instance correctly (line 26), it does not have any effect on the *covered* array of the rules in its replace set. At the end of this Stage, enough information has been gathered to start the next stage, which finalizes the set of rules.

Stage 3: Finalizing Rules At stage 3 (Algorithm 3), the set of rules is finalized. In this Stage, UAC filters the rules based on a greedy method of error reduction. Function *computeError* counts the number of instances that are covered by rule r but have a different class label than $r.class$. The covered instances are then removed from the dataset. Function $addDefaultClass$ finds the most frequent class label among the remaining instances (line 6). In line 8, the number of instances correctly classified by the default class is calculated. $totalError$ is the total errors made by the current rule r and the default class. In fact, each rule with positive coverage over its class, is associated with a particular $totalError$, $defClass$, and $defAcc$ (line 10). After processing the rules, we break the set of rules from the minimum error and assign $default$ and $defApplic$. $defApplic$ is used in rule selection as an estimate of applicability of the default class.

Our rule filtering algorithm has a runtime of $O(|T| \times |R|)$ in the worst case scenario, where $|T|$ is the number of instances in the dataset and $|R|$ is the size

Algorithm 2 UAC Rule Filtering: Stage 2

```
1: RepDAG = ∅
2: for all < i.id, y, ucRule, uwRule >∈ A do
3:    if flagged(uwRule) then
4:       ucRule.covered[y] − −
5:       uwRule.covered[y] + +
6:    else
7:       wSet = allCoverRules(U, i.id, ucRule)
8:       if !RepDAG.contains(ucRule) then
9:          RepDAG.add(ucRule)
10:      end if
11:      for all w ∈ wSet do
12:         w.replace.add(< ucRule, i.id, y >)
13:         w.covered + +
14:         ucRule.incom + +
15:         if !w ∈ RepDAG then
16:            RepDAG.add(w)
17:         end if
18:      end for
19:      Q = Q.add(wSet)
20:   end if
21: end for
22: S ← set of all nodes with no incoming edges
23: while S ≠ ∅ do
24:   r = S.next() {next removes a rule from the set}
25:   for all < ucRule, id, y >∈ r.replace do
26:      if (r.covered[r.class] > 0) then
27:         if id is covered then
28:            r.covered[y] − −
29:         else
30:            ucRule.covered[y] − −
31:            Mark id as covered.
32:         end if
33:      end if
34:      ucRule.incom − −
35:      if ucRule.incom = 0 then
36:         S.add(ucRule)
37:      end if
38:   end for
39: end while
```

of rawSet. The worst case scenario is when at Stage 1, at least one *ucRule* or *uwRule* is the last rule in the sorted rawSet. This case rarely happens because the rules are sorted based on their absolute precedence. UAC also makes slightly more than one pass over the dataset in the rule filtering step. Passes are made in Stage 1 and 2. Note that array A is usually small, given that most of the instances are usually classified by the highest ranked rules. The number of passes is an

important point, because the dataset may be very large. Specially for datasets that can not be loaded into memory at once, it is not efficient to make multiple pases. This is an advantage for UAC over UCBA, which passes over the dataset once for each rule in rawSet. Next section explains the rule selection that is the procedure of classifying test instances based on the set of rules.

Algorithm 3 UAC Rule Filtering: Stage 3

1: $C = \emptyset$
2: **for all** $r \in Q$ **do**
3: **if** $r.covered[r.class] > 0$ **then**
4: $finalSet.add(r)$
5: $ruleErrors+ = computeError(r)$
6: $defClass = addDefaultClass()$
7: $defErrors = computeDefErr(defClass)$
8: $defAcc = addDefAcc(uncovered(D) - defErrors)$
9: $totalError = defErrors + ruleErrors$
10: $C.add(r, totalError, defClass, defAcc)$
11: **end if**
12: **end for**
13: Break C from the rule with minimum error
14: C contains the final set of rules
15: $default = defClass.get(C.size)$
16: $defApplic = \frac{defAcc.get(C.size)}{|T|}$

3.3 Rule Selection

Rule selection is the procedure of classifying a test instance. In the previous sections, excessive rules were filtered out from rawSet. The remaining set of rules is called finalSet and classifies the test instances. UAC selects one classifying rule for each instance. The selected classifying rule has the highest relative precedence with respect to the test instance.

The role of the default class ($default$ in Algorithm 3 line 15) is to reduce the number of rules. The default class predicts the labels of those instances that are not classified by the rules in the finalSet. So the best predicting label for some of the test instances may be default class. But UAC may prefer rules with small PI values to the default class if we follow the procedure of "certain" data classifiers. To prevent this, $defApplic$ is used as an estimate for applicability of the default rule. This value shows the number of training instances that were expected to be classified by the default rule. For example, when two classes, such as a and b, have the same population in the dataset but no rule labeled b exists, default rule has a very important role. Consequently, the value of the default applicability is high. As a result, if the highest precedence rule with respect to a test instance has less applicability than the default rule, the default rule will predict the label for that.

4 Experiments and Results

We use an empirical study to compare UAC against the existing rule based methods. In all of the reported experiments on UAC, the minimum support is set to 1%, the minimum confidence to 0.5 and the maximum number of mined association rules to $80,000$. Each reported number is an average over 10 repetitions of 10-fold cross validations.

Since there is no public repository of uncertain datasets, we synthetically added uncertainty to 28 well known UCI datasets. This method was employed by all the studies in the field including uHARMONY, DTU and uRule, uncertain svm, UCBA, etc. and gives a close estimation of the classifier performance in the real world problems. We selected the same datasets as in [12] to compare our method with the results reported in their paper for uHARMONY, uRule and DTU. This also ensures that we did not choose only the datasets on which our method performs better.

To compare our method against other classifiers, we employ averaging technique and case by case comparison [20]. The same method was employed by many other studies including CBA, uHARMONY, DTU and uRule to prove the better performance of their algorithms. Table 1 provides a comparison between UAC and other existing rule based methods in terms of accuracy. The reported accuracies for uHARMONY (#3), DTU (#4) and uRule (#5) are reproduced from [12]. We applied UAC (#2) to the same datasets generated by the same procedure of adding uncertainty as [12] to make the comparison meaningful. Value N/A, existing in the experiments reported by [12], shows that the classifier has run out of resources in their experiments. In Table 1, uncertainty level is U10@4 meaning that datasets have 10 percent uncertainty, where only four of the attributes with the highest information gain are uncertain. To add a level 10 uncertainty to an attribute, it is attached with a 0.9 probability and the remaining 0.1 is distributed randomly among the other values present in the domain. The accuracies in this table are reported on already discretized versions of the dataset that are available online and referenced in [12].

The accuracies reported show that in most cases UAC has reached higher accuracies. For some datasets the improvement is significantly high, such as *wine* dataset with 36.79% and *bands* dataset with 19.77% improvement over the existing maximum accuracy. UAC reaches higher accuracies on the average too.

We have conducted further extensive experiments comparing UAC and UCBA since both stem from CBA. Due to lack of space we report here only the summary and refer the reader to [21] for further details. Using a new and more general uncertainty model that we propose [21], we compared the accuracy of UAC and UCBA on all 28 datasets as in the previous experiments in Table 1 and show that UAC outperforms UCBA. On average, over the 28 datasets, the accuracy of UAC was 74.7% while UCBA averaged 67.5% if a sampled-based model is used when a numerical attribute is assigned a set of possible values; and respectively 70.3% versus 66.7% if an interval-based model is used [21]. In short, for a numerical attribute, the sampled-based model considers the attribute value to be expressed by a set of values with their respective probabilities, while the interval-

#1 Dataset	#2 UAC	#3 uHAR	#4 DTU	#5 uRule
australian	80.2	**85.37**	83.62	84.35
balance	84.9	**89.3**	56.32	62.88
bands	**78.4**	58.63	N/A	N/A
breast	94.3	65.52	91.27	**94.56**
car	**89.3**	77.72	70.02	70.02
contracep	43.6	47.59	**50.1**	44.26
credit	78.1	**85.95**	84.35	74.35
echo	92	**93.29**	92.37	87.02
flag	45.7	52.42	**59.28**	44.85
german	71.9	69.6	**72.3**	70.1
heart	**77.3**	56.64	53.04	52.39
hepatitis	81.5	**82.52**	80	79.35
horse	72.4	82.88	**85.33**	N/A
monks-1	**99**	91.36	74.64	70.68
monks-2	**75.5**	65.72	65.72	65.72
monks-3	**98.1**	96.4	79.96	68.05
mushroom	**100**	97.45	**100**	99.98
pima	**73.8**	65.11	65.1	67.32
post_oper	58	69.75	**70**	**70**
promoters	66	69	**71.7**	61.32
spect	**81.8**	80.19	79.03	81.65
survival	**74**	73.53	73.53	72.55
ta_eval	**50.4**	45.04	48.34	33.77
tic-tac-toe	**90.8**	76.2	72.65	81.52
vehicle	**69.8**	63.44	64.78	N/A
voting	91.1	92.86	94.48	**94.94**
wine	**87.9**	51.11	42.13	41.57
zoo	**92.3**	88.76	92.08	89.11
Average	**78.5**	74.05	73.04	70.49

Table 1. %Accuracy, reported by rule based classifiers on datasets modeled based on [12] at level of uncertainty of U4@10.

based model considers the attribute value to be an interval with a probability distribution function. Moreover, comparing the training time, UAC was in many cases about 2 orders of magnitude faster (i.e. X100) than UCBA and produced significantly less rules for all tested uncertainty levels. This demonstrates the efficacy of the rule pruning startegy managing to preserve a better set of rules than UCBA.

5 Conclusion

In this paper we propose an effective way to prune associative classification rules in the presence of uncertainty and present a complete associative classifier for uncertain data that encompasses this pruning. Empirical results show that our algorithm outperforms 4 existing rule-based methods in terms of accuracy on average for 28 datasets and also show that UAC outperforms UCBA significantly for these 28 datasets in terms of accuracy even though UAC produces less classification rules and has a smaller runtime than UCBA.

References

1. P. Sen and A. Deshpande, "Representing and querying correlated tuples in probabilistic databases," in *IEEE ICDE*, pp. 596–605, 2007.
2. C. Wang, L.-Y. Yuan, J. H. You, O. R. Zaiane, and J. Pei, "On pruning for top-k ranking in uncertain databases," in *international conference on Very Large Data Bases (VLDB), PVLDB Vol.4, N.10*, 2011.
3. M. A. Cheema, X. Lin, W. Wang, W. Zhang, and J. Pei, "Probabilistic reverse nearest neighbor queries on uncertain data," *IEEE Transactions on Knowledge and Data Engeneering (TKDE)*, vol. 22, pp. 550–564, April 2010.
4. C. C. Aggarwal, Y. Li, J. Wang, and J. Wang, "Frequent pattern mining with uncertain data," in *ACM SIGKDD*, pp. 29–38, 2009.
5. B. Jiang and J. Pei, "Outlier detection on uncertain data: Objects, instances, and inference," in *IEEE ICDE*, 2011.
6. M.-L. Antonie, O. R. Zaiane, R. Holte, "Learning to use a learned model: A two-stage approach to classification," in *IEEE ICDM*, pp. 33–42, 2006.
7. J. Bi and T. Zhang, "Support vector classification with input data uncertainty," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 161–168, 2004.
8. B. Qin, Y. Xia, and F. Li, "DTU: A decision tree for uncertain data," in *PAKDD*, pp. 4–15, 2009.
9. J. Ge, Y. Xia, and C. Nadungodage, "UNN: A neural network for uncertain data classification," in *PAKDD*, pp. 449–460, 2010.
10. B. Qin, Y. Xia, and F. Li, "A bayesian classifier for uncertain data," in *ACM Symposium on Applied Computing*, pp. 1010–1014, 2010.
11. B. Qin, Y. Xia, S. Prabhakar, and Y. Tu, "A rule-based classification algorithm for uncertain data," in *IEEE ICDE*, 2009.
12. C. Gao and J. Wang, "Direct mining of discriminative patterns for classifying uncertain data," in *ACM SIGKDD*, pp. 861–870, 2010.
13. X.Qin, Y. Zhang, X. Li, and Y. Wang, "Associative classifier for uncertain data," in *international conference on Web-age information management (WAIM)*, pp. 692–703, 2010.
14. B. Liu, W. Hsu, and Y. Ma, "Integrating Classification and Association Rule Mining," in *ACM SIGKDD*, pp. 80–86, 1998.
15. O. Zaiane and M.-L. Antonie, "Classifying text documents by associating terms with text categories," *Australasian database conference*, pp. 215–222, January 2002.
16. W. Li, J. Han, and J. Pei, "CMAR: Accurate and efficient classification based on multiple class-association rules," in *IEEE ICDM*, pp. 369–376, 2001.
17. Q. Zhang, F. Li, and K. Yi, "Finding frequent items in probabilistic data," in *ACM SIGMOD*, pp. 819–832, 2008.
18. T. Bernecker, H. p. Kriegel, M. Renz, F. Verhein, and A. Zuefle, "Probabilistic frequent itemset mining in uncertain databases," in *ACM SIGKDD*, 2009.
19. J. R. Quinlan, *C4.5: programs for machine learning*, Morgan Kaufmann Publishers, 1993.
20. J. Demsar, "Statistical comparison of classifiers over multiple data sets", *JMLR*, vol. 7, pp. 1–30, 2010.
21. M. Hooshsadat, Classification and Sequential Pattern Mining From Uncertain Datasets, MSc dissertation, University of Alberta, Edmonton, Alberta, September, 2011.