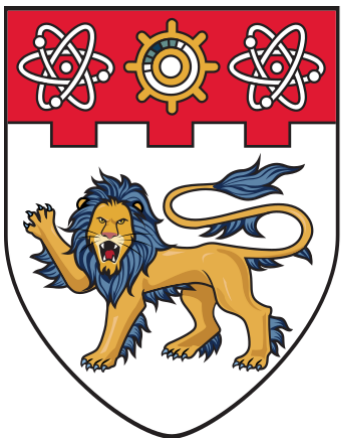
be



NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

CZ4032 Data Analytics and Mining Assignment 2

Name	Matriculation Number
SHALINA SHAM @LIM SHANA	U2022851K
FU XIAOXUAN	U2020847G
LIM YUN HAN, DARREN	U1921275J
YI JIA XIN, JOCELINE	U1920057J
LUMLERTLUKSANACHAI PONGPAKIN	U2023344C

Task 1: Implement an algorithm of Classification based on Association rules.

We referenced a Github code repository (https://github.com/liulizhi1996/CBA) for our implementation of the algorithm, specifically the rmep.py and read.py python scripts. The read.py script reads the dataset and decodes it into a list after pre-processing has been done while the rmep.py script contains the functions for recursive minimal entropy partitioning, used to discretize continuous-valued attributes. This function outputs a list of partition boundaries of the range of continuous-valued attributes in ascending order, allowing us to segment the data.

For mining class association rules, we used the Apriori algorithm. The naive version of the Classification Based on Associations Classifier Builder (CBA-CB M1) algorithm was implemented for the classifier.

Data Pre-processing

For our data pre-processing, we employed 3 steps. We first processed any missing values in the dataset by filling them with the mode of the existing values in the same column, then discretized the numerical attributes using the rmep.py script, and replaced the labels for categorical attributes with a positive integer.

Mining class association rules

The objective of our rule generator is to mine class association rules (CARs) that satisfy the minimum support and minimum confidence of 0.01 and 0.5 respectively. We set the values as such so that rules with a lower support value but high confidence will still be included in the CARs generated. Figure 1 shows the algorithm of our class association rule generator.

Algorithm 1 Class Association Rule Generator

Mine all individual *ruleitems* and calculate their support and confidence by counting the
rulesupCount & *condsupCount*
if *ruleitem*.support > minimum support:
    add this *ruleitem* to frequent *ruleitem*
if the *ruleitem* in f_1 satisfies the minimum support and minimum confidence:
    generate rules from f_1 and store in CAR_1
prune CAR_1
append pruned CAR_1 to CARs
For (k=2; f_{k-1} is not None: k++):
    generate the candidate *ruleitems* by joining *frequent ruleitems* in f_{k-1} together
    for *ruleitem* in candidate *ruleitems*:
      if ruleitem.support > minimum support:
          add this *ruleitem* to f_k
    if the *ruleitem* in f_k satisfies the minimum support and minimum confidence:
      generate rules from f_k and store in CAR_k
    prune CAR_k
    append pruned CAR_k to CARs

Figure 1: Pseudocode of Class Association Rule Generator

The algorithm scans through the entire dataset once, mines all frequent rules items and proceeds to generate the class association rules (CARs). If the rule item satisfies the minimum support and minimum confidence, which is declared before running, the rule item is stored. From there we prune the rules and append it to the CARs.

In our implementation, created a class, apriori, to generate the CAR with the following methods:

Pass – this method runs through the dataset to count the number of rule items and returns a dictionary with the corresponding rule's support and confidence count.

Counters – this method is used to count the support and confidence of the candidate rule items and returns both the support and confidence counts.

Car_candidate_gen – this method takes in the class label of the data, a set of tuples of the items and a list of candidate rules, and returns the list of candidate rules.

Expand – This method creates new candidate rule items from the frequent rule items from *Pass* and returns the candidate rule items as a set.

Search – this method counts the occurrences of the rule items with and without the target label, and returns the candidate rule items with their support and confidence counts.

Prune – this method prunes the CAR based on the minimum support and minimum confidence declared and returns a dictionary of pruned rule items with their corresponding support and confidence counts.

Add_rule – this method adds the rule passed into the function to the set of CAR and returns True if rule is added.

Run – this method generates the CAR by calling the methods and returns it in the form of a set.

Classifier

We built our classifier in accordance with the M1 classifier specified in the paper by Liu et al. [1]. The objective of our classifier is to choose rules with higher priority from our CAR to cover the entire dataset. Figure 2 shows the pseudocode of the CBA-CB M1 algorithm.

Algorithm 1 M1 Classifier Building

```
1:  cars_list = precedence sort (cars);
2:  for each rulecase ∈ cars_list do
3:    temp =    ;
4:    mark = False;
5:    for each datacase ∈ dataset do
6:      is_satisfy_value = does datacase satisfy rulecase;
7:      if is_satisfy_value is not None then
10:       store index of datacase in temp;
11:       if is_satisfy_value is True then
12:         mark = True; // the rulecase is marked correct for correctly classifying the datacase
13:       end
14:     end
15:   end
16:   if rulecase is marked (mark == True) then
17:     remove all rows from dataset whose index is in temp;
18:     // these 3 functions are defined in the insert method of the Classifier class
19:     append rulecase to classifier's rule list;
20:     select default class for classifier;
21:     compute total no. of errors;
22:   end
23: end
24: // the following two functions are performed in the discard method of the Classifier class
25: Find the 1st rule in Classifier with lowest total no. of errors and drop all rules after that in Classifier;
26: Append default class label of this rule to end of Classifier;
27: return classifier;
```

Figure 2: Pseudocode of the CBA-CB M1 Algorithm

The classifier builder involves three steps:

- Step 1: Sort the class association rules according to its priority. Rule A has priority over rule B if rule A has a higher confidence than rule B. If the confidence of both rules are equal, rule A has priority if the support for rule A is greater than that of rule B. If both support and confidence are equal for both rules, the rule generated earlier has priority.
- Step 2: Select rule for classifier from the sorted sequence. In this step, errors are computed as well.
- Step 3: Discard rules that do not help improve the classifier's accuracy.

Task 2: Classification results

Task 2 requires us to test our algorithm on multiple datasets, 5 of which are used in the paper by Liu et al. [1] with the same setting, and 3 other datasets were self-sourced.

We decided to use the Iris dataset, Wine dataset, Australian dataset, Waveform dataset, and Pima dataset from the paper. These datasets were chosen from the paper as they provided a good range of error rates in the paper. The paper used 26 datasets for comparison and came to an average error rate of 17.1 with discretization. We chose 2 datasets with error rates that fall below the average (Iris and Wine dataset), 2 with a higher error rate (Pima and Waveform datasets) and 1 with an error rate similar to the average value (Australian dataset).

The 3 other datasets we chose were the Tic-tac-toe dataset, the Car dataset and the Breast cancer dataset, obtained from the UCI Machine Learning Portal. These datasets were chosen as they had 2 categories, possessing only 2 class labels. Furthermore, while most datasets chosen from the paper possess mostly continuous numerical attributes, these 3 datasets contain mostly categorical attributes. Thus, by choosing these 3 datasets, we are able to get a better sensing of how our algorithm will perform across different types of datasets.

Dataset description

Figure 3 highlights the key features of our chosen datasets.

Dataset	No. of Attributes	No. of Numerical Attributes	No. of Categorical Attributes	No. of Class labels	No. of Instances
Iris	4	4	0	3	150
Wine	13	13	0	3	178
Australian	14	6	8	2	690
Waveform	21	21	0	3	5000
Pima	8	8	0	2	768
Tic tac toe	9	0	9	2	958
Car	6	0	6	2	1728
Breast Cancer	9	0	9	2	286

Figure 3: Features of chosen datasets

Classification Results

Figure 4 below shows the results of our implemented CBA-CB M1 classifier against the 8 chosen datasets.

Dataset	Accuracy w/ pruning	No. of CARs w/ pruning	No. of Rules for Classifier
Iris	94.67%	104	7
Wine	95.88%	561	82
Australian	90.31%	52555	1119
Waveform	83.08%	14375	600
Pima	80.33%	3634	101
Tic tac toe	92.32%	15545	28
Car	75.93%	1086	41
Breast cancer	87.38%	8793	76
Average	87.49%	12081.63	256.88

Figure 4: Classification results of CBA-CB M1 algorithm

Task 3: Other Classification Methods

Task 3 requires us to apply other classification methods to our datasets. The decided datasets are the same datasets in task 2 which are Iris dataset, Wine dataset, Australian dataset, Waveform dataset, Pima dataset, Tic-tac-toe dataset, Car dataset, and Breast cancer dataset. The classification methods we applied are Decision trees, Random forest, Neuron networks, and SVM.

For neuron networks, we applied 2 hidden layers with ReLU activation function and an output layer with softmax activation function. For each dataset, we applied random search to find the optimized number of neuron units of each layer and the learning rate. For the rest classification methods, we applied the default value of those methods.

Classification Results

Figure 5 shows the accuracy of the classification methods against chosen datasets

Dataset	Decision trees	Random forest	Neuron networks	SVM
Iris	97.78	97.78	97.78	97.78
Wine	94.44	98.15	87.04	77.78
Australian	79.71	85.51	78.74	65.22
Waveform	77.20	85.33	83.80	86.93
Pima	72.29	75.76	71.86	75.32
Tic tac toe	92.01	95.49	94.79	86.11
Car	97.50	97.30	100	90.94
Breast cancer	75.58	75.58	63.95	75.58
Average	84.10	88.86	84.75	81.96

Figure 5: Accuracy of classification methods

Task 4: Improvement of the CBA Algorithm

Task 4 requires an improved version of the CBA algorithm to be implemented. In order to improve the accuracy of classification results, Quantitative CBA (QCBA) was implemented [2]. QCBA is an algorithm that is designed as an extension of CBA with more emphasis on post-processing the rules comprising the CBA classifier, used to recover information lost during discretisation of attributes. In the post-processing phases, the fit of each rule on the training data is improved and more types of rule pruning

are performed to reduce the number of rules. The implementation of QCBA was inspired by PyArc. The various algorithms used in post-processing are explained below along with the pseudo-codes.

QCBA first generates a rule list using CBA algorithm, based on training data that has been discretized. For discretisation, we used the MDLP library provided in Python which uses entropy based discretisation methods [3].

Algorithm 1 shows how the rule list is optimised through refitting each rule, while allowing the literals in the antecedent of the rules to have smaller boundaries that correspond to unique attribute values in the data. Figure 1 shows the pseudocode of Algorithm 1.

Algorithm 1 Refit Rule *refit()*

Require:
r - input rule generated using discretized training data
Ensure:
rule r with refit literals

```
for each rule in rules do
    for each literal in rule.IfComponent do
        Compute the range of discrete values ∈ literal
        Update the range of the literal in the IfComponent
    end for
end for
return the refitted rules
```

Figure 6: QCBA Algorithm 1

Algorithm 2 prunes literals from each rule. A literal is considered redundant if the removal of the literal does not result in a drop in rule confidence.

Algorithm 2 Prune literals from rule IfComponent

Require:
r - input rule
Ensure:
rule r with redundant literals is removed

```
for each rule in rules do
    while a literal is removed from the rule IfComponent do
        for each literal in r.IfComponent do
            r'.IfComponent = rule.IfComponent without literal
            calculate the confidence of r'
            if r'.conf > r.conf then
                r.IfComponent = r'.IfComponent
            end if
        end for
    end while
end for
return the pruned rule list
```

Figure 7: QCBA Algorithm 2

Algorithm 3 trims the boundary of the literals, which is similar to algorithm 1. This removes any values that are misclassified by rule r.

Algorithm 3 Rule Trimming

Require:
r - input rule
Ensure:
rule r with trimmed literals

```
for each rule r in rules do
    Compute the rows covered by each rule
    for each literal  r.IfComponent do
        Compute the min and max for r
        Update the interval with range(min,max)
    end for
end for
return the updated rule list
```

Figure 8: QCBA Algorithm 3

Algorithm 4 extends the ranges of literals in each rule, but the extension is adopted only if the rule confidence is improved.

Algorithm 4 Extend range of literals

Require:
r - input rule
Ensure:
extended rule r

```
for each rule in rules do
    for each literal in rule.IfComponent do
        Extend range of literal to form rule r'
        Calculate the confidence of r'
        if r'.conf > r.conf and r'.support > r.support then
            update the literal of r
        end if
    end for
end for
return rule list
```

Figure 9: QCBA Algorithm 4

While the previous algorithms have adjusted and improved the coverage of individual rules, algorithm 5 reduces rule number through post pruning of rules. Before any rule is pruned, the rules are sorted according to criteria in CBA.

1. Rule 1 is ranked higher if it has higher confidence than rule 2.
2. Rule 1 is ranked higher if it has the same confidence as rule 2, but greater support than rule 2.
3. Rule 1 is ranked higher if it has shorter antecedent (i.e. fewer conditions) than rule 2.

During post pruning, a default rule is appended to the list of rules and each rule is then tested against the data. If a rule wrongly classifies any object, it is removed. Objects that satisfy the rule are also discarded. This helps to determine the rule with the fewest number of errors.

Algorithm 5 PostPruning of extended rules

Require:
extended rule list
Ensure:
some elements of the rule list are removed, default rule added

```
Sort the rules based on rule ranking criteria in CBA
for each rule r in rules do
    if r misclassifies one object then
        Remove r
    end if
    Calculate the default class from CBA algorithm
    Update the default class based on the cut off rule
end for
return rule list
```

Figure 10: QCBA Algorithm 5

Algorithm 6 prunes overlapping rules by iterating through all the rules that classify the same classes as the default rule. The rules that overlap with the default rule are potential rules to be removed. A rule is removed if its removal does not affect the classification results.

Algorithm 6 Rule Overlap Pruning

Require:
rule list
Ensure:
some elements of the rule list are removed

```
defaultRule = last rule in rules
Iterate rule list backwards until r.ThenComponent != defaultRule.ThenComponent
if r.ThenComponent == defaultRule.ThenComponent then
    Remove the rule
end if
return rule list
```

Figure 11: QCBA Algorithm 6

Results and Analysis of QCBA Algorithm

Dataset	Accuracy (CBA)	Accuracy (QCBA)
Iris	94.67%	99.33%
Wine	95.88%	100%
Australian	90.31%	90.16%
Waveform	83.08%	70.58%
Pima	80.33%	83.98%
Tic tac toe	92.32%	100%
Car	75.93%	74.17%
Breast cancer	87.38%	98.24%
Average	87.49%	89.56%

Figure 12: QCBA Algorithm Results

From Figure 12 we can observe and compare the accuracies of the CBA and QCBA algorithms on the 8 selected datasets. Our QCBA implementation gives better results for almost all of the chosen

datasets, as compared to the CBA results. Among the 8 datasets, *iris, breast cancer*, *tic-tac-toe* and *wine* datasets observed a significantly higher accuracy for QCBA, whereas *pima* observed a slightly higher accuracy. For the *australian* and *car* datasets, the accuracies obtained from CBA and QCBA are almost the same. However, similar to the results mentioned in the QCBA paper, the implementation of QCBA algorithm does not always guarantee better results than CBA algorithm. For *waveform* dataset, the accuracies of QCBA algorithm are lower than CBA. In terms of the win-tie-loss metric, QCBA surpasses CBA: QCBA wins on 5 datasets, CBA wins on 1 dataset and there is a draw on 2 datasets.

References

[1] Bing Liu, Wynne Hsu, and Yiming Ma. 1998. Integrating classification and association rule mining. In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98). AAAI Press, 80–86.

[2] Kliegr, Tomas. "QCBA: Postoptimization of quantitative attributes in classifiers based on association rules." arXiv preprint arXiv:1711.10166 (2017).

[3] Fayyad, Usama, and Keki Irani. "Multi-interval discretization of continuous-valued attributes for classification learning." (1993).

Individual Contribution Claims

LIM YUN HAN, DARREN: Task 1 and 2
YI JIA XIN, JOCELINE: Task 1 and 2
LUMLERTLUKSANACHAI PONGPAKIN: Task 3
FU XIAOXUAN: Task 4
SHALINA SHAM @LIM SHANA: Task 4