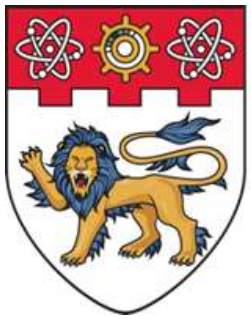


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

School of Computer Science and Engineering

CZ4032 - Data Analytics and Mining

Group Assignment

Done By:

Tiviatis

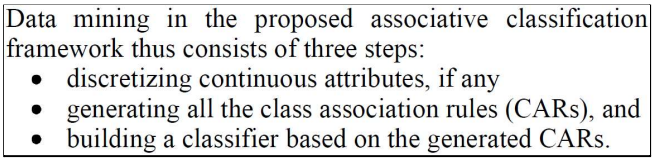
Inbanathan Vaishnavi

Kusalavan Kirubhaharini

Quek Lin Hui

Classification Based on Associations (CBA)

Mining Class Association used: an Apriori-based rule generator
Classifier used: M2 Classification Building Algorithm


Data mining in the proposed associative classification framework thus consists of three steps:

- discretizing continuous attributes, if any
- generating all the class association rules (CARs), and
- building a classifier based on the generated CARs.

Figure 1

The proposed framework as seen in Figure 1 was derived from the paper. The implementation of the code thus adheres to these steps. The main .py file containing the algorithm is cba.py, and it calls functions from .py files contained in the *"Mine_Classi_Alg"* and *"Structure"* folders.

Our code is derived from the source code of a module named pyARC[1]. Instead of using the module itself by importing the functions, we used portions of their source code[2], such as their data structures portion on the creation of a transactional database (found in the *"Structure"* folder). We also modified their CAR generation and classification algorithm to suit the M2 algorithm which we are using ("*generating_CARS.py*" and "*m2classi.py*" can be found in the *"Mine_Classi_Alg"* folder). We used Orange[3] to discretize our data.

The CAR generation process finds the complete set of CAR with its antecedent, consequent, support, confidence and id (as defined in "*car.py*" in *"Structure"* folder), and finds frequent rule items that fulfil minimum support. The CARS generation code("*generating_CARS.py*") uses fim.apriori (pyFIM[4]) in method "*generateCARS*". Rules are thus generated with Apriori after data has been converted to a transactional database.

We have selected the M2 algorithm for building the classifier due to its good performance and linear scale up. In the M2 algorithm, we are able to find the best rule in the ruleset to cover each case, making it more efficient than the M1 algorithm. In the M2 algorithm. only slightly more than 1 pass is made over the dataset (1 in the initial stage and subsequently to find all rules that make a wrong classification), making it much faster than the M1 algorithm due to fewer passes.

The following is a breakdown of our implementation of the M2 algorithm, we took the pseudocode from *Integrating Classification and Association Rule Mining* as reference and implemented the algorithm accordingly.

Stage 1: Correct and Wrong Rule Identification

We start by identifying the rules with the highest precedences using the function maxCoverRule that takes in the arguments of a sorted set of rules and the data instance. We group that into either cRule (rules that correctly classifies data) or wRule (rules that wrongly

[1] https://pypi.org/project/pyarc/
[2] https://github.com/jirifilip/pyARC
[3] https://docs.biolab.si/orange/2/reference/rst/Orange.feature.discretization.html
[4] https://borgelt.net/pyfim.html

classified data) *(lines 2-4 according to figure 2 below).* If the rule is found within the data case, the rule will be set as the cRule if there is no existing cRule and the rule correctly classifies the data. This is similar for wRule where the rule would be set as wRule if there is no existing wRule and the rule wrongly classifies the data. Should there be a cRule with highest precedence, we will then add the cRule to a set of all correctly classified rules (U) *(line 5)* and update the number of class cases covered *(line 6)*.

Should the cRule have higher precedence than the wRule . *(line 7)* We will add it into Q, a set of all cRules with higher precedence than their corresponding rules *(line 8)* and mark the cRule accordingly to indicate that it classifies the case correctly *(line 9)*. Otherwise, if the rule has the higher precedence, we would append the information to set A (the list of structures that have wrule with higher precedence than cRule) *(line 10)*. We create set A to set up the algorithm for Stage 2.

```
1   Q = ∅; U = ∅; A = ∅;
2   for each case d ∈ D do
3       cRule = maxCoverRule(C_c, d);
4       wRule = maxCoverRule(C_w, d);
5       U = U ∪ {cRule};
6       cRule.classCasesCovered[d.class]++;
7       if cRule ≻ wRule then
8           Q = Q ∪ {cRule};
9           mark cRule;
10      else A = A ∪ <d.id, d.class, cRule, wRule>
11  end
```

Figure 2: Pseudocode for Stage 1 of M2 Algorithm

Stage 2: Considering the wrongly classified records

Firstly, iterate through set A and check whether wrule is marked (i.e. it is the cRule of at least one case and wRule would cover the datacase) *(line 2)*. Update the rule count of the rules that cRule and wRule covers to account for this *(line 3-4)*.

If the wRule if not marked, identify rules that wrongly classifies the datacase and have higher precedence than the crules found in set U. The function would return the rules with higher precedences that may replace cRule to cover the case *(line 5-8)*. Should the replacing rule have higher precedence than the cRule , the replacing rule would be able to “override” the desired cRule that corresponds to the wRule . As such, we will add the replacing rule to the set wset, which contains the rules with higher precedence than the cRule *(line 10).*

```
1   for each entry <dID, y, cRule, wRule> ∈ A do
2     if wRule is marked then
3       cRule.classCasesCovered[y]--;
4       wRule.classCasesCovered[y]++;
5     else wSet = allCoverRules(U, dID.case, cRule);
6          for each rule w ∈ wSet do
7            w.replace = w.replace ∪ {<cRule, dID, y>};
8            w.classCasesCovered[y]++;
9          end
10         Q = Q ∪ wSet
11    end
12  end
```

Figure 3: Pseudocode for Stage 2 of M2 Algorithm

Stage 3: Determine the default classes and the total error count

Firstly, sort the set Q (set of cRules with higher precedence than their corresponding wRules that correctly classifies at least one record) according to decreasing order of precedence *(line 3).* Reiterate through the set of "overriding" rules *(line 4)*. If the rule has already been covered by a previous rule, we will decrease the distribution array value of the current rule *(line 8)*. Otherwise, if the rule has not been previously referred to by another rule, we will decrement the array value of the overriding rule instead *(line 9)*.

```
1   classDistr = compClassDistri(D);
2   ruleErrors = 0;
3   Q = sort(Q);
4   for each rule r in Q in sequence do
5     if r.classCasesCovered[r.class] ≠ 0 then
6       for each entry <rul, dID, y> in r.replace do
7         if the dID case has been covered by a
              previous r then
8           r.classCasesCovered[y]--;
9         else rul.classCasesCovered[y]--;
10      ruleErrors = ruleErrors + errorsOfRule(r);
11      classDistr = update(r, classDistr);
12      defaultClass = selectDefault(classDistr);
13      defaultErrors = defErr(defaultClass, classDistr);
14      totalErrors = ruleErrors + defaultErrors;
15      Insert <r, default-class, totalErrors> at end of C
16    end
17  end
18  Find the first rule p in C with the lowest totalErrors,
      and then discard all the rules after p from C;
19  Add the default class associated with p to end of C;
20  Return C without totalErrors and default-class;
```

Figure 4: Pseudocode for Stage 3 of M2 Algorithm

Next, count the accumulated number of wrong classifications that would result if the current rule were to be the last rule in the classifier *(line 10)* and update the distribution array to exclude the number of records covered by the current rule *(line 11)*. We set the majority class displayed by the array at that point of time as the defaultClass *(line 12)*.

Based on the defaultClass, we determine the respective defaultError (i.e. number of errors that will result if all unsatisfied records are allocated from the defaultclass) *(line 13)*. The totalError for the current rule would be the sum of the defaultError and ruleErrors *(line 14).* Append the totalError into the totalError array *(line 15)*.

With the list of total errors for each rule, we are able to find the rule with the least total error and that would be considered the strongest cRule *(line 18)*. Add the strongest cRule into the final classifier. We add all the rules in the rulelist up to and including the above mentioned rule as well as the default rule that produces the default class associated with the identified rule*(line 19)*. These rules would form the final classifier.

Classification Results

Dataset description:

1. Iris[5] - A dataset used to predict the class of an iris plant from 3 different classes.
2. Waveform[6] - A dataset used to predict the class of a wave from 3 different classes.
3. Breast Cancer[7] - A dataset to predict whether there would be a recurrence event of breast cancer.
4. German[8] - German credit dataset to predict whether a customer is "good" (i.e. suitable for a loan) or "bad" (i.e. unsuitable for a loan)
5. Heart[9] - A dataset to predict the absence or presence of heart disease.

[5] https://archive.ics.uci.edu/ml/datasets/iris
[6] https://archive.ics.uci.edu/ml/machine-learning-databases/waveform/
[7] https://archive.ics.uci.edu/ml/datasets/breast+cancer
[8] https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/
[9] https://archive.ics.uci.edu/ml/datasets/statlog+(heart)

6. Stroke[10] - A dataset to predict whether a patient would experience a stroke event.
7. University Placement[11] - A dataset containing the placement of students into industries. We are using it to predict whether students are “placed” (i.e. employed) or “unplaced” (i.e. unemployed)

For our comparison, we chose to compare the accuracy across the different classifiers for the different datasets.

The following table shows the comparison of accuracy scores of test datasets:

Dataset/ Classifier used	Neural Network (NN)	Support Vector Machine (SVM)	Decision Tree (DT)	Random Forest Regression (RF)	CBA
Iris	0.93	0.93	0.93	0.93	0.90
Wave	0.84	0.88	0.73	0.85	0.764
Breast Cancer	0.63	0.72	0.70	0.72	0.649
German	0.70	0.73	0.70	0.80	0.70
Heart	0.81	0.76	0.76	0.78	0.815
Stroke	0.95	0.96	0.92	0.96	0.950
University Student Placement	0.86	0.88	0.76	0.86	0.767

Table 1: Accuracy comparison of CBA with other classification methods

From Table 1 above, CBA only performs relatively better than the other classifiers for the heart dataset. For all other datasets, the performance of the CBA algorithm is comparable to the other algorithms despite being on the lower end. All algorithms use train-test split with an 80:20 ratio on random state 25 with the same parameter settings across different datasets for consistency.

Quantitative Classification By Associations (QCBA)

CBA does not work well on quantitative data as it causes information to be lost during the discretization process. QCBA[12] is an improvised version of CBA, where it uses input rules from CBA to obtain an optimized set of rules. It improves on CBA as it recovers information lost in CBA’s discretization via postprocessing CBA generated rules. This helps to minimize the information lost from discretization of quantitative continuous data, thereby improving the accuracy of the classification. It also provides new rule pruning techniques to reduce the size of the classifier. This helps to speed up the post-processing.

[10] https://www.kaggle.com/fedesoriano/stroke-prediction-dataset
[11] https://www.kaggle.com/benroshan/factors-affecting-campus-placement
[12] https://arxiv.org/pdf/1711.10166.pdf

The QCBA algorithm consists of the following steps according to Figure 5 below:

Algorithm 1 QCBA *qcba()*

Input: *rules* – input rule list generated by CBA
Output: optimized *rules*

1: *rules* ← remove any rules with empty antecedent from *rules*.
2: **for** *rule* ∈ *rules* **do** {process rules in the sort order given by Fig. 3}
3: *rule* ← *refit*(*rule*) {cf. Alg. 2}
4: *rule* ← *pruneAttributes*(*rule*){cf. Alg. 3}
5: *rule* ← *trim*(*rule*){cf. Alg. 4}
6: *rule* ← *extendRule*(*rule*) {cf. Alg. 5, 6}
7: **end for**
8: *rules* ← *postprune*(*rules*) {cf. Alg. 7, postpruning adds a new default rule, if postpruning is disabled, QCBA ensures that default rule is added at this point.}
9: *rules* ← *drop*(*rules*) {cf. Alg. 8 for transaction-based version and Alg. 9 for range-based version of default rule overlap pruning}
10: **return** *rules*

Figure 5: QCBA Algorithm

An example of a set of rules generated using QCBA is shown below in Figure 6:

id	antecedent	consequent	conf	supp
1	Humidity=(80;100]	→ Class=1	0.11	0.80
2	Temperature=(30;35]	→ Class=4	0.14	0.63
3	Temperature=(25;30] and Humidity=(40;60]	→ Class=4	0.08	0.60
4	Temperature=(15;20]	→ Class=2	0.11	0.57
5	Temperature=(25;30]	→ Class=4	0.14	0.50
6		→ Class=2	0.28	0.28

Figure 6: Rules generated by CBA rule generator

Each rule in the set of rules generated using the CBA rule generator will be processed in the following order given below. The rule with id 3 from the table above is used as an example here to explain the process.

A literal is defined as (A,V) where A is an attribute and V is the range value.

[For simplicity, let Temperature be denoted as A and Humidity as B.]

1. Refitting of rules
    - The input rule is (A (25,30] and B (80,100])→ Class 4
    - For each literal, the values V' that correspond to the value of that literal from the whole dataset is computed.
    - The intersection values between V' and V are then obtained to get the minimum (min) and maximum (max) of this intersection.
    - The value range V for attribute A is then set to [min,max]
    - The output rule will be ((A,[26,30]) and (B,[85,100])) → Class 4
2. Pruning attributes
    - The Input rule is ((A,[26,30]) and (B,[85,100])) → Class 4. This is the output from the previous algorithm (refitting).
    - The first literal is removed from the rule. The new rule now becomes (B,[85,100]) → Class 4.
    - If the confidence of (B → Class 4) is greater than confidence of (A & B → Class), the rule is replaced by B → Class instead.

- This process is repeated for every literal.
- Loop is broken only under 2 scenarios:
    i. The confidence of all subsets is better than the superset
    ii. The confidence of no subset is better than the superset
- The output will be (B,[85,100]) → Class 4.

3. Trimming boundaries
    - The input for this algorithm is the output for the pruning algorithm (B,[85,100]) → Class 4.
    - The training instances R which are correctly classified by this rule are obtained. This consists of the instances satisfied by all literals in this rule.
    - For every literal, the training instances L which are correctly classified by that literal are also obtained.
    - Distinct values of attribute A from the instances L are obtained.
    - If the number of distinct values is lesser than or equal to 1, the loop is continued with another literal instead.
    - Else, if there are at least 2 distinct values for A in L, we get the distinct values of A from training instances R
    - The min and max of these distinct values are obtained and the new value V for this attribute A is assigned to be [min,max].
    - An example output will be (B,[85,90]) → Class 4
    - This process trims the boundaries and narrows the range of the literals in the rule.
4. Extending rule
    - The Input for this algorithm is (B,[85,90]) → Class 4
    - The literal *l: (B,[85,90])*
    - For every literal l, the endpoints are extended using the getExtensions() function. The lower and higher extensions are obtained
        i. *Lower extension, l_L: (B,[82,90])*
        ii. *Higher extension, l_H: (B,[85,91])*
    - The combined final extension is $l = l_L \cup l_H$ = *(B,[82,91])*
    - The extension output will be (B,[82,91]) → Class 4.
    - This output is only accepted only if it satisfies the crisp accept or conditional accept condition as shown in the Figure 7 below.

Inputs:

1. $\Delta_{conf} = confidence(\text{refined rule}) - confidence(\text{original rule})$
2. $\Delta_{supp} = support(\text{refined rule}) - support(\text{original rule})$

Crisp accept: rule is accepted if confidence improves at least by $minImprovement$ and support of the rule does not drop below the original rule on the input of extension:

1. IF $\Delta_{conf} \geq minImprovement$ **and** $\Delta_{supp} \geq 0$ then **true**
2. ELSE **false**

Conditional accept: rule is conditionally accepted if confidence improves at least by $minCondImprovement$:

1. IF $\Delta_{conf} \geq minCondImprovement$ then **true**
2. ELSE **false**

Figure 7: Criteria for crisp and conditional accept

- If multiple extension rules meet the crisp accept criteria, the extension rules are sorted according to criteria depicted in Figure 8 below.

1. rule A is ranked higher if confidence of rule A is greater than that of rule B,
2. rule A is ranked higher if confidence of rule A is the same as confidence of rule B, but support of rule A is greater than that of rule B,
3. rule A is ranked higher if rule A has a shorter antecedent (fewer conditions) than rule B.

Figure 8: Criteria for rule sorting

- The current rule is replaced by the first ranked new extension rule and this rule undergoes further extensions. The candidates of the current rule go through many acceptance stages. This direct extension process terminates when the current candidate rule does not result in a crisp acceptance. The current candidate then undergoes the conditional acceptance process. If it is not conditionally accepted, this candidate is dropped and the next candidate is checked instead. If it is conditionally accepted, it goes through the crisp accept stage or the conditional accept stage again. This process is repeated until the extension is unsuccessful.

5. Post-pruning
   - The entire resulting ruleset then undergoes post-pruning.
   - This is similar to the CBA algorithm described earlier.
6. Overlap pruning
   - Range-based
     i. If the antecedent of the current rule and any other rule are the same but the consequent classes are different, then the lower ranked rule will be pruned.

Additional investigations were also done by altering the initial CBA rule generator used before implementing qcba to a top-k rule generator, where the CARs generation code("*generating_CARS.py*") replaces fim.apriori with fim.arules in method "*top_rules*". The output gives the top k rules, where k=1000 across all datasets for consistency. The code is rather simple and iteratively increases minimum support, and maximum length, while decreasing minimum confidence as a secondary priority, until all 1000 rules get filled or when all iterations are exhausted. These generated CARs are then thrown into the M2 algorithm and then qcba. The performance of original CBA vs QCBA vs QCBA (top k) is found in *Table 2* below.

Classifier used	CBA		QCBA		QCBA (top k)	
Dataset	Accuracy	Runtime (s)	Accuracy	Runtime (s)	Accuracy	Runtime (s)
Iris	0.90	0.0280	0.933	0.110	0.933	0.159
Wave	0.764	1.38	0.768	12.4	0.81	14.4
Breast Cancer	0.649	0.208	0.807	1.69	0.789	1.19
German	0.7	17.4	0.745	32.5	0.76	31.0

Heart	0.815	0.301	0.926	2.45	0.870	1.45
Stroke	0.950	4.28	0.951	14.3	0.951	16.6
University Student Placement	0.767	0.249	0.907	2.50	0.860	1.91

Table 2: Comparison of runtime and accuracy between CBA, QCBA and QCBA(top k)

Both QCBA and QCBA(top k) consistently provide better performance than CBA in terms of accuracy across all the datasets. However, this comes at the expense of increased runtime. The implementation of the top k rule mining allowed QCBA to achieve even better accuracies in the Wave and German datasets. However, the other sets may not have improved as much due to non optimal k-values used(k=1000).

References:

Liu, B., Hsu, W., & Ma, Y. (1998). *Integrating Classification and Association Rule Mining - AAAI*. Retrieved October 20, 2021, from https://www.aaai.org/Papers/KDD/1998/KDD98-012.pdf.

Coenen, F. (2005). THE LUCS-KDD IMPLEMENTATIONS OF THE CBA ALGORITHM. Retrieved October 20, 2021, from https://cgi.csc.liv.ac.uk/~frans/KDD/Software/CBA/cba.html.

Dannina, H. (2020, January 28). *Underrated Machine Learning Algorithms - apriori*. Medium. Retrieved October 20, 2021, from https://towardsdatascience.com/underrated-machine-learning-algorithms-apriori-1b1d7a8b7bc.

Chonyy. (2020, October 31). *Apriori: Association rule mining in-depth explanation and python implementation*. Medium. Retrieved October 20, 2021, from https://towardsdatascience.com/apriori-association-rule-mining-explanation-and-python-implementation-290b42afdfc6.

Bhavesh. (2020, December 15). *Iris data prediction using decision tree algorithm*. Medium. Retrieved October 20, 2021, from https://medium.com/analytics-vidhya/iris-data-prediction-using-decision-tree-algorithm-7948fb68201b.

Wilkinson, P. (2020, November 16). *Introduction to decision tree classifiers from scikit-learn*. Medium. Retrieved October 20, 2021, from https://towardsdatascience.com/introduction-to-decision-tree-classifiers-from-scikit-learn-32cd5d23f4d.

Kliegr, T. (2019, October 18). *QCBA: Postoptimization of Quantitative attributes in classifiers based on Association rules*. QCBA: Postoptimization of Quantitative Attributes in Classifiers based on Association Rules – arXiv Vanity. Retrieved October 20, 2021, from https://www.arxiv-vanity.com/papers/1711.10166/.

Li, Wenmin & Han, Jiawei & Pei, Jian. (2001). CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules. Proceedings - IEEE International Conference on Data Mining, ICDM. 369-376. 10.1109/ICDM.2001.989541.

Cong, Gao & Tan, Lee & Tung, Anthony & Xu, Xin. (2005). Mining Top-k Covering Rule Groups for Gene Expression Data.. Proceedings of the ACM SIGMOD International Conference on Management of Data. 670-681. 10.1145/1066157.1066234.