

School of Computer Science and Engineering

CZ4032 - Data Analytics and Mining

Group Assignment

Done By:

***2) Implement an algorithm of Classification based on Association rules. You can implement any algorithm in the paper, or a variant of these algorithms.***

***Your coding should have two parts:***

***• Mining Class association rules***

***• Building a classifier***

***You can refer to the online code, and reuse part of the code. But you must understand the code. If you reuse some online coding, you need to detail what code you use in your report. In the report, you can also explain the pseudocode of building classifier in your implementation.***

***One reference code:***

[***https://cgi.csc.liv.ac.uk/~frans/KDD/Software/CBA/cba.html***](https://cgi.csc.liv.ac.uk/~frans/KDD/Software/CBA/cba.html)

***PART 2 (3pg)***

Implementation from: Part 1 paper on CBA (Classification Based on Associations)

Mining Class Association used: an Apriori-based rule generator

Classifier used: Classifier builder called M2

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]](#footnote-0). Instead of using the module itself by importing the functions, we used portions of their source code, such as their data structures portion on the creation of a transactional database and discretizing attributes(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)

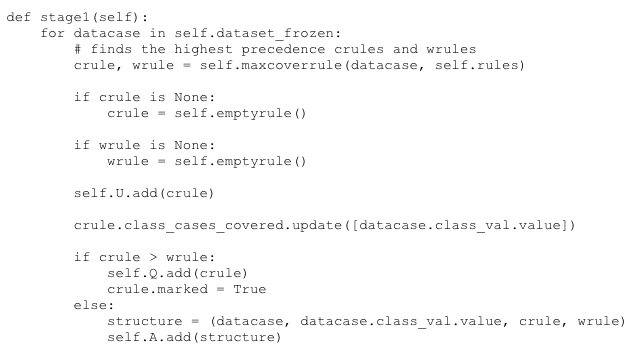
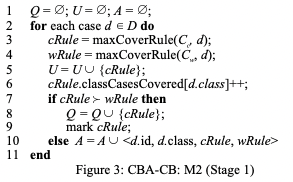
The CAR generation process finds the complete set of CAR with its antecedent, consequent, support, confidence and id (as defined in “*car.py*” in *“Structure”* folder), and finds frequent rule items that fulfil minimum support. The CARS generation code(“*generating\_CARS.py*”) uses fim.apriori (pyFIM[[2]](#footnote-1)) in method “*generateCARS*”. Rules are thus generated with Apriori after data has been converted to a transactional database. An alternative is to find top k rules using fim.arules and create CARs using only those rules.

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, which would require to make a pass over the remaining data for each rule (cite the JA). 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. This is supported by the experiment conducted in the article where M2 generally had much better performance than the M1 Algorithm.

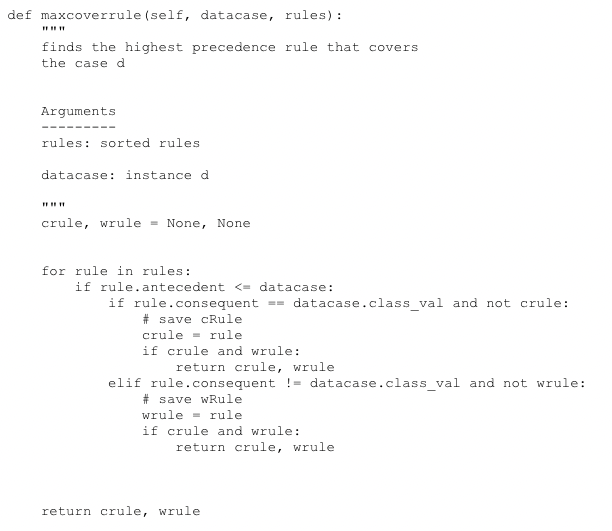
(Not sure whether need to put in the screenshot or not)

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. These rules are either crule (correctly classifies data) or wrule (wrongly classifies data). We identify the rules with the highest precedences using the function maxCoverRule as shown below.

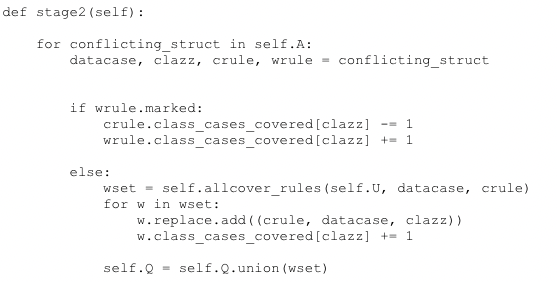
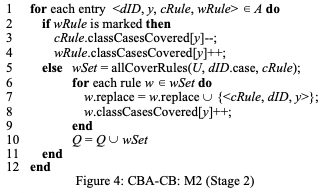


The function maxCoverRule takes in the arguments of a sorted set of rules and the data instance. If the rule is found within the datacase, the rule will be set as the crule if there is no existing crule and the rule correctly classifies the data. This is similar for wrule where the rule would be set as wrule if there is no existing wrule and the rule wrongly classifies the data. As this is an ordered set of rules, the rules that are returned would be those of the highest precedence.

Should there be a crule with highest precedence, we will then add the crule to a set of all correctly classified rules (U).

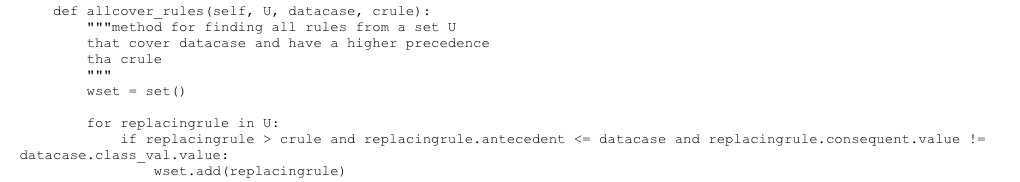
We update the number of class cases covered and should the crule have higher precedence than the wrule. We will then add it into Q, a set of all crules with higher precedences than their corresponding rules and mark the crule accordingly to indicate that it is classifying the case correctly. Otherwise, we would append the set Q with the information.

Stage 2: Considering the wrongly classified records



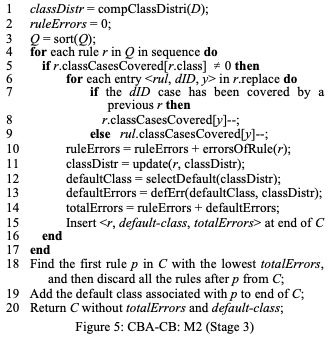
Stage 2 of M2 Algorithm

In step 2, we iterate through the list of structures that have wrule with higher precedence than crule. We check whether wrule is marked as it would mean that it is the crule of at least one case and wrule would cover the case of index “datacase”. We would then update the rule count of the rules that crule and wrule covers to account for this. With the function allcover\_rules(), we 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. The algorithm is as follows:



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 precedences than the crule.

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



Firstly, we sort the set Q (set of crules with higher precedences than their corresponding wrules that correctly classifies at least one record) according to decreasing order of precedence. We begin to reiterate through the rules that were overriding first. If the rule has already been covered by a previous rule, we will decrease the distribution array value of the current rule. Otherwise, if the rule has not been previously referred by another rule, we will decrement the array value of the overriding rule.

Next, we will count the accumulated number of wrong classification that would result in if the current rule were to be the last rule in the classifier and update the distribution array to exclude the number of records covered by the current rule. We set the majority class displayed by the array at that point of time as the defaultClass.

Based on the defaultClass, we determine the respective defaultError, which would be the number of errors that will result if all unsatisfied records are allocated from the defaultclass. We sum up the ruleErrors and the defaultError to find the totalError for the current rule and we append it into the totalError array

Upon getting the list of total errors for each rule, we find the rule with the least total error and that would be considered the strongest crule. We then add it 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. These rules would form the final classifier.

**3) Use 5 datasets used in the KDD’98 paper with the same setting to conduct classification tasks using the code you develop in Part 2. Report the classification results of your software in Part 2. You can choose any 5 datasets. These datasets can be obtained from the UCI machine learning portal (https://archive.ics.uci.edu/ml/datasets.php).**

**In addition, please choose another 2 datasets from the portal or other datasets you are interested (e.g., data in Kaggle) to report results. To make it easier, you are suggested to choose datasets with 2 categories.**

**In the report, you need to document the dataset description, and the classification results (similar as those report in the paper.**

***PART 3(1.5 page)***

**4) Find open softwares for other classification methods: Decision trees, random forest, and SVM. Compare with them in terms of classification accuracy. You can use measures such as F-score.**

**In the report, you need to document the classification results of different method**

***PART 4 (.5)***

**5) Advanced part: Design and implement an algorithm to improve the accuracy of classification results by the software that you develop in part 2. You can use the methods in the following papers. You can also design other improvement. You need to evaluate your improved algorithms on the same set of datasets you used in part 3, and compare with other methods on the accuracy of classification.**

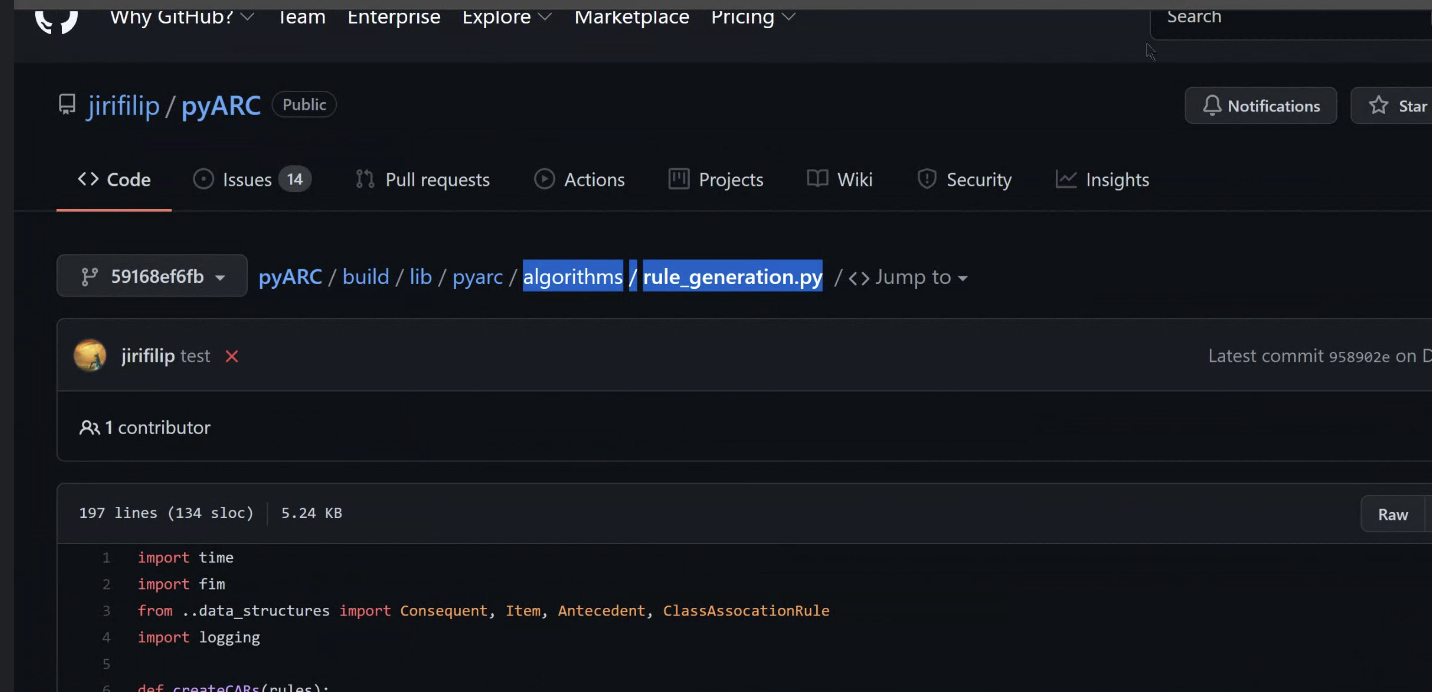
**In the report, you need to explain the algorithm you design for part 5. You can give pseudocode and explain it line by line. You also need to document the classification results of your algorithm.**

***PART 5(2pg)***

<https://towardsdatascience.com/underrated-machine-learning-algorithms-apriori-1b1d7a8b7bc>

<https://towardsdatascience.com/apriori-association-rule-mining-explanation-and-python-implementation-290b42afdfc6>

[https://github.com/jirifilip/pyAR](https://github.com/jirifilip/pyARC)C



Ignore top\_rules fn

* Get rules (apriori -> CARS fn) then use lift

Questions

* M2 algorithm on apriori
* Apriori why do we need to arrange them afterwards
* Is the precedence of the association rules different for apriori and classification with respect to m2 algorithms
* Clarify what he means by having 2 classes

Timeline (to do before meeting)

* Complete respective Q4 before next meeting
* Part 3 can try playing around to get accuracies
* Try part 5 by replacing apriori with eclat

Post meeting

* REfine the code and alter it
* Work on the explanation and report
* finish the video and submit

Datasets

* iris
* waveform
* breast cancer
* heart
* German

2 other datasets (2 classes?)



1. https://pypi.org/project/pyarc/ [↑](#footnote-ref-0)
2. https://borgelt.net/pyfim.html [↑](#footnote-ref-1)