NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4032 Data Analytics and Mining

Project 1

Group 45

Group Members:

Lee Cheng Han (U1920206L)
Ryan Phuay Qian Hao (U1922240H)
Koh Yi Joshua (U1922368F)
Nur Lydia Afiqah Binte Rozali (U1922339C)

1. Introduction **3**

2. Classification algorithm implementation based on Association rules **3**
2.1 Algorithm selection 3
2.2 Algorithm pseudocode 3

3. Datasets and Classifications **5**
3.1 Fundamental Methodology 5
3.1.1 Data Selection 5
3.1.2 Data cleaning and preprocessing 5
3.1.3 Data transformation 5
3.1.4 Data mining 5
3.2 Result documentation 5

4. Classification methods comparisons **6**
4.1 Decision trees 6
4.2 Random forest 6
4.3 Support Vector Machines (SVM) 6
4.4 Classification Accuracies 7

5. Accuracy improvement **7**
5.1 Algorithm methodology 7
5.2 Algorithm pseudocode 8

1. Introduction

This project is implemented using Python Jupyter Notebook. Pruning is not adopted in this project. Run time is based on a 64-bit PC with 16GB RAM on Windows OS.

2. Classification algorithm implementation based on Association rules

2.1 Algorithm selection

We have decided on implementing the Apriori algorithm. The codes for the algorithm were referenced from ***pyArc***, which is an implementation of the CBA algorithm based on Association algorithms. We also incorporated several external reference codes as well.

2.2 Algorithm pseudocode

From the reference code, we implemented the standard Apriori algorithm to complete the first portion of task 2, which was to mine Class Association Rules (CARs) to generate our list of rules.

CBA-RG(RULE GENERATION)

---

count items and class occurrences
generate CARs
CARs is subject to pruning operation (optional)

```
1       F = {large 1-ruleitems};
2       cars = genRules(F);
3       PrunedCARs = Pruning.prune(cars);
5       for items in F:
6               C   = candidateGen(F);
7               for datacase in database:
8                       C = ruleSubset(C  , d);
9               for candidate in C:
10                      c.condsupCount++;
11                      if C == F:
12                              c.rulesupCount++;
13              end
14      end
```

Frequent *ruleitems* founds in the *K-1th* pass are used to generate candidate *ruleitems* **C**
Scan the database and updates the support counts of candidates in **C**
After new frequent *ruleitems* have been identified to form **F**, the algorithm produces the rules CAR using the *genRules* function
Rule pruning is performed

After that was complete, in order to satisfy the second part of the task, we used the M1Algorithm referenced from KDD98-012 to build a classifier. This algorithm was selected as the datasets that we have elected to work on do not exceed any memory thresholds, allowing us to make use of the naive algorithm.

M1 ALGORITHM (CBA-CB(Classifier building))

In essence:
We need to choose a set of high precedence rules in our CARs which are able to cover our dataset.

```
1       Sort generated rules;
2        for rule in sorted_rules:
3               for datacase in database:
4                       if rule correctly classifies datacase, mark rule;
5               if rule is marked:
6                       insert rule into classifier C;
7                       Remove cases the rule covers from dataset;
8                       Default class selection from majority class in remaining;
9                       Record number of errors made by the classifier & default class;
10              end
11      end
```

Find and mark the first rule with the least number of errors, *rule*[i];
Discard rules after this *rule*[i];
Final classifier is *rules*[0:i];
return *rules*[0:i] as the classifier;

1) The generated rules are sorted to ensure we will choose the highest precedence rules for the classifier (34)
2) Rules are selected from the set of generated rules in order of the sorted sequence. For every rule, the dataset is combed to find cases that are covered by the rule (70). If the rule correctly classifies one case, it is marked (74-83).
3) If a rule has at least one mark, it is considered as a potential rule in our classifier (87).
4) The cases it covers are then removed from the dataset (92-94), and a default class is selected(98-120).

5) The number of errors made by the current classifier is computed and recorded(129-145).
6) When there are no rules or training cases left, the rule selection process is completed.
7) Discard rules in our classifier which do not improve the accuracy of the classifier.
8) The first rule at which there is the least number of errors recorded on the dataset is the cutoff rule (151-155).
9) Discard all rules after this cutoff rule as they will only cause more errors (155-157).
10) The undiscarded rules and default class of the last rule in the classifier forms our final classifier (161-164).
11) Return the classifier (179).

3. Datasets and Classifications

3.1 Fundamental Methodology

3.1.1 Data Selection

The datasets used in this project were obtained from UCI and Kaggle; most are used in the KDD '98 paper. The datasets used are: Iris, Ionosphere, Mushroom, Tic-Tac-Toe, Zoo, Mental Health and Hepatitis.

3.1.2 Data cleaning and preprocessing

After selecting and understanding the datasets, data cleaning and preprocessing is done to remove any missing values, duplicated and incorrectly formated data. This is to improve data quality and increase overall productivity of the data.

3.1.3 Data transformation

During data transformation, one-hot encoding was used for columns such as Gender to convert it to a categorical column. Binary values of '0's and '1's are assigned to these columns.

3.1.4 Data mining

Data mining is then done to learn and do predictions.

3.2 Result documentation

Dataset Name	CBA	No.of CARs	Run-time CBA_RG	Run-time CBA_CB (M1)	No.of rules in C

Iris	79.3	1325	8s	7s	37
Ionosphere	87	62207	15s	53s	21
Mushroom	61.5	115574	63s	293s	31
Tic-Tac-Toe	99.5	9450	31s	113s	8
Zoo	85.7	53896	30s	115s	7
Mental Health	86	97998	75s	353s	185
Hepatitis	65.3	168055	45s	181s	34
Average	80.6	57786	38s	159	46

Fig 1

4. Classification methods comparisons

Three classification methods and their variations are implemented to classify the 7 datasets mentioned in this document. Decision Trees, Random Forest and Support Vector Machine (SVM) are used. The classifiers used are imported from Sklearn. The split ratio of the testing and training data is set to 80:20. The n-estimator for Random Forest is set to 60 and SVM is set to linear.

4.1 Decision trees

Decision Trees is a classification model that is used to predict data and to classify them in the form of a tree structure. Data is splitted recursively into nodes based on rules (classification features).

4.2 Random forest

Random Forest is a supervised learning technique which classifies similarly to Decision Trees but without overfitting. The dataset is divided into subsets and merges multiple decision trees into 1.

4.3 Support Vector Machines (SVM)

Support Vector Machines (SVM) is one of the most robust prediction methods. It is a linear model which performs classification by drawing a line between classes which automatically avoids overfitting data.

4.4 Classification Accuracies

Figure 2 shows the accuracy rate for classification as well as the F1-Score (Macro Avg) of 7 datasets. Random Forest has the best accuracy rate and F1-score as compared to Decision Trees and SVM. The number and complexity of the data are some factors affecting the accuracy. Understanding which algorithm or methods for the right problem is important. In the case of the Zoo and Tic-Tac-Toe dataset, a complex or a dataset with a lot of classes would not be suited for SVM. Decision Trees can handle datasets with more classes but it would not be suited for big datasets.

	Decision Tree		Random Forest		SVM	
Dataset Name	Accuracy Rate	F1-Score	Accuracy Rate	F1-Score	Accuracy Rate	F1-Score
Ionosphere	0.86	0.84	0.93	0.92	0.87	0.86
Iris	1	1	1	1	1	1
Mushroom	1	1	1	1	0.97	0.97
Tic-Tac-Toe	0.88	0.86	0.93	0.92	0.65	0.39
Zoo	0.95	0.71	0.95	0.80	0.95	0.80
Mental Health	0.73	0.72	0.83	0.83	0.84	0.78
Hepatitis	0.58	0.43	0.70	0.50	0.71	0.56

Fig 2

5. Accuracy improvement

5.1 Algorithm methodology

For this section, after exploring some other methods, we had decided to try improving our algorithm by making use of a method called Quantitative Classification Based Association algorithm, which is an improved version of traditional CBA. As in traditional CBA we revert the original attribute space to “edit”discovered association rules, refining the scope of literals (conditions, attribute-value pairs) in the antecedent of the rules. This then results in the fit of the rules within the data to improve, rendering some of the rules and attributes redundant. These can be removed, making the resulting classifier smaller and thus more efficient. This is done by the QCBA algorithm which has a total of 6 post algorithms to refine the classifications.

The QCBA is separated into 3 parts. The first is the CBA implementation which was what we had done in task 2. The second is the tuning of individual rules which includes the refitting of the rules, literal pruning, trimming and extension. Lastly, to end the process off, we perform pruning of the optimized rule list, involving post pruning and rule overlap pruning.
In the first part, we will improve the fit of the individual rules to the training data. This increases the coverage of the individual rules and reduces their length by removing redundant attributes and rules. In the second phase, three types of rule pruning are performed to reduce the number of rules.

5.2 Algorithm pseudocode

```
refit()
for literal = (A, V ) in antecedent(r): {V is a value range, e.g. interval [30;30) and A is attribute, e.g. humidity}
        A =all unique values appearing in training data in attribute A.
        left =min(A ∩ V)
        right =max(A ∩ V)
        r=replace literal in r with new literal = (A,[left,right])
end
return r

literalPruning()
attrRemoved =FALSE
While(attrRemoved =TRUE)
        for literal  in antecedent(r)do {Literals are iterated in arbitrary order}
        r′=remove literal from r
        if confidence( r′) >= confidence(r) t:
                R=r′
                attrRemoved =TRUE
                Break
        Else
                attrRemoved =FALSE

        end
Return

trim()
CCBR =training instances covered and correctly classified by r
for literal = (A, V ) in antecedent(r):
        CCBL =training instances covered by literal
        distValsL =distinct values training instances have in attribute A
        if size(distValsL) <=1:
                continue
        end if
```

```
        distValsLinR =distinct values of attribute A in CCBR
        V′=[min(distValsLinR), max(distValsLinR)]
        r=replace literal in r with new literal = (A, V ′)
end
return r

postPruning()
cutoffRule = NULL
cutoffClass =most frequent class in T
lowestTotalError = |T| − |t in T:class(t) = cutoffClass|
totalErrorsWithoutDefault =0
rules =sort rules according to criteria in Fig. 1
defClass =most frequent class in T
{Data coverage pruning}
for all r in rules do
        covered =instances in T matched by antecedent(r)
        corrCovered =instances in T matched by antecedent(r) and consequent(r)
        T = T\covered {Remove instances covered by r from training data}
        if corrCovered=NULL:
                rules =rules \r {remove r from rules}
        else
                misclassified =covered -corrCovered
                totalErrorsWithoutDefault =totalErrorsWithoutDefault +misclassified
                defClass =most frequent class in T
                defaultRuleError = |T| − |t in T:class(t) = defClass |
                totalErrorWithDefault =defaultRuleError +totalErrorsWithoutDefault
                if totalErrorWithDefault<lowestTotalError:
                        cutoffRule, lowestTotalError, cutoffClass =r, totalError WithDefault,
                defClass
end
        end
end
{Default rule pruning}
rules =remove all rules below cutoffRule from rules
rules =append new default rule “{} → cutoffClass’ at the end of rules
return rules
```

Dataset	CBA (Accuracy)	QCBA (Accuracy)	CBA (Time)	QCBA (Time)
Ionosphere	0.873	0.892	44s	89s

Iris	0.897	0.998	8s	9s
Mushroom	0.613	0.877	391s	452s
Tic Tac Toe	0.995	0.998	23s	56s
Zoo	0.706	0.825	55s	93s
Mental Health	0.869	0.885	121s	199s
Hepatitis	0.654	0.917	29s	72s

For the first step, QCBA refits the boundaries of the rule to a finer grid, which corresponds to unique attribute values actually appearing in the data.
Next, the literal pruning step removes literals (attribute-value pairs) whose removal does not decrease the rule confidence from rules.
Then, with the trimming operation literal boundaries are shaved off for values that belong solely to instances that are misclassified by r.

Afterwards, the ranges of literals in the body of each rule are attempted to be enlarged. The range of each literal is increased one literal and one boundary at a time. The extension is generally accepted only if it improves rule confidence. To overcome local minima, the extension process can provisionally accept drops in confidence.

Now, the number of rules can be reduced using the adaptation of CBA's data coverage and default rule pruning. This will also add a default rule to the end of the rule list. This is considered as the process of postpruning. Each rule is then matched against the training data and if a rule does not correctly classify any object, it is discarded. If not, the rule is kept. The data coverage pruning is then combined with the default rule pruning, determining the rule with the lowest number of errors on the training data if rules below it are replaced by a default rule, and performs this replacement.

Overlap pruning iterates through all rules classifying into the same class as the default rule. These rules all overlap with the default rule both in terms of coverage and class assigned and are thus candidates for pruning. They can be removed only if their removal will not change classificationof instances in training data or in the entire instance space they correctly classify by rules that are between them and the default rule. We consider two versions of overlap pruning: transaction-based and range-based.The transaction-based version removes rules if there are no transactions in the training data, which would be misclassified as a result of the removal.The range-based version analyzes overlaps in the range of literals between the pruning candidate and all the potentially clashing rules below it. The pruning is confirmed only if the potentially clashing rules cover different areas. Range-based pruning thus guarantees a solution that generalizes beyond the training data.Its potential disadvantage is that it removes less rules, since it is stronger than the transaction-based pruning.