Simon Fraser University

Assignment 4 - CMPT 741 — Data Mining, Fall 2019

Date**: November 20th, 2019**

Name: **Anurag Bejju**
Student ID: **301369375**

1.

1.1. Algorithm [1]

```
$F_1 = \{large\ 1ruleitems\}$ ; # counts the item and class occurrences to determine the frequent 1- ruleitems
$CAR_1 = genrules(F1)$; # From this set of 1-ruleitems, a set of CARs (called $CAR_1$ ) is generated by genRules
$prCAR1 = pruneRules(CAR_1)$; # $CAR_1$ is subjected to a pruning operation
$for\ k = 2;\ F_{k-1} \neq \emptyset\ ; k++)\ do$
    # frequent ruleitems $F_{k-1}$ found in the $k-1$th pass are used to generate the candidate ruleitems $C_k$ using
    the condidateGen function
     $C_k = candidateGen(F_{k-1})$;

     # It then scans the database and updates various support counts of the candidates in $C_k$
     $for\ each\ data\ case\ d \in D\ do$
          $C_d = ruleSubset\ (C_k, d)$ ;
          $for\ each\ candidate\ d \in C_d\ do$
               $c.condsupCount++$;
               $if\ d.class = c.class\ then\ c.rulesupCount++$
          $end$
     $end$
     # After those new frequent ruleitems have been identified to form $F_k$
     $F_k = \{ c \in C_k \mid c.rulesupCount >= minsup\}$

     $CAR_k = genRules(F_k)$; # the algorithm then produces the rules $CAR_k$ using the genRules function
     $prCAR_k = pruneRules(CAR_k)$; # rule pruning is performed on these rules
$end$
$CARs = \cup_k CAR_k$ ; # The final set of class association rules is in CARs
$prCAR_k = \cup_k prCAR_k$ ; # Those remaining rules after pruning are in $prCAR_k$
```

__pruneRules__ : The function pruneRules uses the pessimistic error rate based pruning method. The process is as follows:

- *If rule r's pessimistic error rate is higher than the pessimistic error rate of rule r^- (obtained by deleting one condition from the conditions of r), then rule r is pruned.*
- *This pruning can cut down the number of rules generated substantially*

As you can see in the above provided algorithm for *Classification Based on Associations – Rule Generation Algorithm* (called CBA-RG), it is very similar to the apriori algorithm. I have coloured the similarities with *green* and dissimilarities with *red*. The *ruleSubset* function that takes a set of candidate ruleitems C_k and a data case d to find all the ruleitems in C_k whose condsets are supported by d as well as the operations related to *support counts of the candidates in* C_k are similar to those in the Apriori algorithm. The only difference is, we would be incrementing the support counts of the condset and the ruleitem separately whereas in algorithm Apriori only one count is updated. This allows us to compute the confidence of the ruleitem .

1.2. Design Explanation [1]

If given a *test record* (t_i), represented as itemset Y (only feature items, no class itemGiven) and a set of classification rules (R), we can classify this test record using a very simple algorithm as follows. We will first *use CBA Classifier Builder Algorithm (M2)* that takes a set of classification rules (R) and returns a set of high precedence rules in R to cover D. This classifier is in the format $< r_1, r_2, r_3, r_4 \dots r_n,\ default\ class >$ where $r_i \epsilon R\ and\ r_a > r_b\ if\ b > a.$. Default class is the majority class after all the rules that applied are removed. Once we have these rules with their order, we will try to find the first rule that satisfies the test case and classify it accordingly. If no rule applies to the case, it takes the default class.

Algorithm

```
Input:
    t_i : test record
    R : set of classification rules

Output:
    c_{t_i} : predicted class for the test record

Algorithm:

#CBA_Classifier_Builder_M2: takes a set of classification rules (R) and returns a set of high precedence rules in R to cover D. This classifier is in the format < r_1, r_2, r_3, r_4 ... r_n, default class > where r_i ϵ R and r_a > r_b if b > a. Default class is the majority class after all the rules that applied are removed.

C = CBA_Classifier_Builder_M2(R)
Classifed = False   # Classified flag

# Iterate over all the rules based on their precedence inside the classifer. Assign the class of the very first rule that satisfies its conditions
For every rule r_i in C < r_1, r_2, r_3, r_4 ... r_n >:
    if t_i satisfies the conditions of r_i then:
        Classifed = True
        return class of c_{t_i} where c_{t_i} is the class for rule r_i
if Classifed == False:

    # assign the default class if t_i did not satisfy any conditions of the rules < r_1, r_2, r_3, r_4 ... r_n >
    return c_{t_i} where c_{t_i} is the default class in C
```

2.

2.1.

Given a social network represented as an (undirected) graph $G = (V, E)$, where V denotes the set of nodes (representing actors) and E denotes the set of edges (representing relationships between the two connected actors), we can perform the task to detect communities in the social network, i.e. clusters in the graph using " *Girvan-Newman betweeness method for graph partition - [Newman and Girvan, 2004]* *[2]* " algorithm. This is a divisive hierarchical clustering method used to detect communities in complex systems based on the concept of edge betweenness. The betweenness of an edge E in graph G is defined as the number of pairs for which the shortest path goes through E.

Design: [2]

This proposed algorithm detects communities by progressively removing edges that are most likely "between" communities from the original network. It uses the concept of vertex betweenness (i.e the total count of shortest paths between pairs of nodes that run through it) as an indicator of highly central nodes in networks.

We will be finding the "*edge betweenness*" of an edge (*the number of shortest paths between pairs of nodes*) that run along it. In case there is more than one shortest path between a pair of nodes, each path is assigned equal weight such that the total weight of all of the paths is equal to 1. If a network contains communities or groups that are only loosely connected by a few inter-group edges, then all shortest paths between different communities must go along one of these few edges. Thus, the edges connecting communities will have high edge betweenness (at least one of them). By removing these edges, the groups are separated from one another and so the underlying community structure of the network is revealed.

The end result of the Girvan–Newman algorithm is a *dendrogram*. As the Girvan–Newman algorithm runs, the dendrogram is produced from the top down (i.e. the network splits up into different communities with the successive removal of links). The leaves of the dendrogram are individual nodes. We can assess this dendogram and decide a cutoff measure for the clusters.

2.2. Pseudo Code [2]

The algorithm proposed by Newman and Girvan for identifying communities is as follows:

```
First calculate the betweenness for all edges in the network.
Until $E \rightarrow \phi$ do:    #do until all edges are removed
    Remove the edge with the highest betweenness.
    recalculate betweennesses for all edges affected by the removal.
```

Run time: Worst Case: $O(m^2 n)$.

The betweennesses is calculated by using the fast algorithm of Newman [2], which calculates betweenness for all m edges in a graph of n vertices in time $O(mn)$. Because this calculation has to be repeated once for the removal of each edge, the entire algorithm runs in worst-case time $O(m^2 n)$. However, after the removal of each edge, we only have to recalculate the betweennesses of those edges that were affected by the removal, which is at most only those in the same component as the removed edge. This means that running time may be better than worst-case for networks with strong community structure (those that rapidly break up into separate components after the first few iterations of the algorithm). We have can't calculate the betweennesses of all edges only once and then remove them in order of decreasing betweenness, because if two communities are connected by more than one edge, then there is no guarantee that all of those edges will have high betweenness—we only know that at least one of them will. By recalculating betweennesses after the removal of each edge we ensure that at least one of the remaining edges between two communities will always have a high value.

References:

[1] *Integrating Classification and Association Rule Mining* - Liu, Bing and Hsu, Wynne and Ma, Yiming - Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining - KDD'98
[2] *Community structure in social and biological networks* - M. Girvan, M. E. J. Newman - Proceedings of the National Academy of Sciences Jun 2002, 99 (12) 7821-7826; DOI: 10.1073/pnas.122653799