A GENETIC ALGORITHM BASED ASSOCIATION RULE LEARNING SYSTEM

PETER EBEAR, DAVID ENGEL, EVAL LOUGHLIN, JAMES MACISAAC, AND SHANE SIMS

1. Introduction

The task of this paper is to satisfy the Winter 2018 CPSC 599.44 Midterm Report requirements, and in so doing, present the proposal for the learning system that we indent to implement. To This end, we will present a sketch of our intended learning system in the following pages. The structure of this paper is as follows. We first present our learning method by means of introducing the key parts of our proposed system. We then describe how we intend to include relevant knowledge not already in the dataset into the learning system. Finally we describe how we intend to deal with the requirement of being able to read a list of already known association rules, and how these will be used to prevent our learning system from learning these rules again.

2. Learning Method

For the task of learning association rules from Statistics Canada's *1999 to 2014 National Collision Database*, we will implement a learning system whose learning system is based on a Genetic Algorithm. Our reasoning for this choice, and why we have not chosen to build a system that uses the *apriori* algorithm is as follows:

- Given the size of our dataset (over 6 million examples), we believe that the intractability of using the *apriori* algorithm with i-itemsets when $i \in \mathcal{Z}^+$ becomes large, makes it unsuitable for finding interesting association rules. This of course assumes that there exists interesting association rules with a large number of clauses on either of both sides of the implication. While the *apriori* algorithm requires i iterations over the dataset, the element of randomness inherent to a well designed genetic algorithm will allow our learning system to move around the search space in a non-breadth-first manor, potentially increasing the probability that novel association rules are found.

  Need to verify/fact check this entire item
- By using a genetic algorithm as the basis of our learning system, it is conceptually clear how we will be able to meet the requirement that our system be able to use a list of already known rules to prevent these rules from being learned again. Moreover, by using a genetic algorithm, our system will be able to use this list to prevent repeated learning of these rules in a more intelligent way than simply filtering out already known rules during post processing. Our method for doing this is described in section 4 below.

- The ability to define our own fitness function provides an additional way to add knowledge to our learning system. Instead of using just coverage and accuracy as a measure of fitness, we might also add additional terms to our fitness function as a means of directing the search to areas that we wish to explore; for example by specifying the consequent, or a mandatory part of the antecedent. A further treatment of this idea is given in 3 below.

  Not only can we adjust weights in our fitness function to direct our learning system to different areas of the association rule search space, but we can also add elements to our fitness function to

Having provided our justification for choosing to build a genetic algorithm based learning system, a high-level description of the key parts of our system will provide clarity on our intentions.

2.1. **Individual Representation.** An individual in our system will be representative of an association rule. We will represent individuals of a given state in our learning system according to the following high level schematic:

There might be a better way to represent individuals, but note that any ideas we have should be compatible with the genetic operations

Status	Status	...	Status	Status
C_YEAR	C_MNTH	...	P_SAFE	P_USER

Naturally this representation requires further comment:

- 'Status' can hold a value of either 0, 1, or 2. This represents if a feature is included in the rule, part of the antecedent, or part of the consequent, respectively.
- We do not include a field for the C_CASE field, as it is an incident identifier [cite data dictionary], and thus not meaningful in this context.
- As none of the features have a negative number as an allowable value, we will represent null with -1.
- Allowable (but non-integer) values such as NN,UU, etc., will be mapped to unique negative numbers, so that type remains consistent in the implementation.

This representation of an individual corresponds to a single association rule. It includes a index for every possible feature of an arbitrary example in our dataset; its presence and role in the given rule indicated by the 'Status' number. This representation allows for simple implementation of genetic operators (described in section 5), as will be demonstrated with an example in the final section of this paper. With this representation scheme in mind, we would represent the association rule $(C\_WTHR = 1) \Rightarrow (C\_SEV = 2)$ as:

0	0	0	0	**2**	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
-1	-1	-1	-1	**2**	-1	-1	-1	**1**	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Of course there are a number of ways that this representation of an individual can be implemented. Most obviously this can be implemented as a vector, in the form as an array of small (ex. primitive type 'short' in Java) integer pairs, each index corresponding to a feature, with the pair at this location holding the status number

and feature value respectively. If sparseness of these individuals becomes a concern for memory efficiency, we can also look at storing the individual as a pairs of small integers in a hash table, with their respective indices serving as a key. Consider the following as an example of how we could implement individuals as an array of arrays, using the above rule as the individual:

$$[[0,-1],[0,-1],[0,-1],[0,-1],[2,2],[0,-1],[0,-1],[0,-1],[1,1],[0,-1],\ldots,[0,-1]]$$

2.2. **Initial Population.** The initial population for our learning system will consist of m individuals. The non-null features of a given individual will be chosen randomly so that even weight is given to rules rules of all sizes. For our purposes, the minimum rule size will be that with one feature in the consequent and one (different) feature in the antecedent. The value of each included feature in a given rule will also be chosen randomly from among the allowed values (as determined in the data dictionary). The value of m will be determined empirically once our system is implemented.

2.3. **Genetic Operators.** Our system will utilize the two usual genetic operators, namely crossover and mutation. How each of these will operate on an individual(s) selected for the operation in our system, is outlined as follows:

should we include a section describing our control?

- $Crossover(rule_1, rule_2)$: This binary operation works by choosing a random number r between 1 and 20 (recall that there are 22 features in our data items). This number r then forms the crossover point of the operation. Then indices 0 to r of $rule_1$ will be prepended to indices $r+1$ to 21 of $rule_2$.
- $Mutation(rule)$: This unary operation requires three random numbers to be generated. First we choose a random number r_1 between 0 and 21. This corresponds to the index of our mutation point. We then randomly choose r_2 to be either 1 or 2. Let s be the current value (0, 1, or 2) of 'status' at index r_1 of $rule$. We randomly generate another number, r_2 as either 1 or 2. Next we compute $s' \equiv s + r_2 \pmod 3$; forcing the status bit to change while allowing for all either of the possible alternative values with the same probability. Finally, if $s' \in \{1, 2\}$, we generate r_3 as a random value in the domain of the feature corresponding to r_1, and we set the pair at index r_1 in $rule$ to be (s', r_3). Otherwise, we set 'status' to 0.

2.4. **Fitness Function.** Next we describe the basic version of our fitness function. Note that the next section provides a way that this basic structure can be enhanced to add additional knowledge to the system. Let $|D|$ be the number of items in our dataset. The basic version of our fitness function is composed of two terms, which are themselves functions:

$$fitness(rule) = \begin{cases} 0 & \text{if } rule \text{ has clauses in antecedent and consequent} \\ (\frac{coverage(rule)}{|D|} + accuracy(rule)) \times 100 & \text{otherwise} \end{cases}$$

Here *coverage* and *accuracy* are computed in the regular ways.

3. Adding Knowledge

We propose the following ? ways of adding knowledge not already contained in our dataset, to our learning system:

James mentioned some ideas for this. I think this will be where any interesting new rules are learned and we should probably focus some energy on making a creative list of ideas here

(1) Expanding our dataset to include ?
(2) Accepting command line arguments specifying clauses that must be present in the antecedent and/or consequent of any learned rule. This would provide us with a means to explore certain areas of the association rule search space as guided by the knowledge of the operator at runtime.
(3) Adding a new term to our fitness function that captures a measure of how "interesting" a given association rule is. One way of doing this would be to check if the rule is in a list of known AR rules produced by WEKA's *apriori* algorithm, and reducing a rules fitness score if this is the case. Alternatively, we might compute the hamming distance between the given rule and the top 10 of those produced by WEKA, and assign a higher fitness for those rules with a higher mean distance.

how else might we quantify "interesting"?

4. Detecting Applicable Knowledge

The requirement that our system be able to read a list of already known rules, and avoid producing those same rules throughout the course of learning, is relatively simple when using a genetic algorithm based learner. Our system fulfill this requirement by cross checking this list upon the creation of any new individual (be it during the creation of the initial population, or upon applying a genetic operator), and discarding any that is found to already exist in this list.

Should we be providing algorithm details on how this works? I.e. how we will parse the list, etc. ?

5. Example Association Rule Learning

(TODO: the example should show a couple of generations, and exhibit both genetic operators - using a simplified dataset (reduced features and examples))

We should work these our ASAP as it will probably reveal some shortcomings in our proposed system. It would also be a good idea for everyone to do a hand example, to get a solid idea of how the system works. This should help during implementation time.