CS 590D - Homework #1

Verykios, Vassilios

March 19, 1998

Problem 4 (i)

In order to start the data mining process we need to pre-process the data in such a way that can be given as input to the Progol system. Even if the structural information is not contained in the file itself, we know that is the attribute that is related with each one of the columns from the file that contains information about the car database. The way we have followed to incorporate this information into our data mining algorithm is to create appropriate predicates that are representative of the attributes at hand and also of the degree the attribute is represented for each one of the cars. More information about the specific procedure we are using will be given in the next part. In the sense that a specific column corresponds to a well defined problem attribute, we could say that this is absolutely irrelevant for the data minimg process. What it really matters is to use whatever symbolic information we can get out of the problem specification in order to be able afterwards (that is after the algorithm has terminated) to interpret the results. Even though some of the attributes are numeric, we should say first that they are integer values , and then that there is only a very small number of them. Thus in the first place, this should not cause too much trouble in our modeling. In our approach we have created appropriate symbolic identifiers that correspond to the name of the numeric value (i.e., $2 \longrightarrow two$). Still, we could have used them as placeholders, in the sense that we did not accept by the system to treat them as ordinal values since the system does not know that 2 is greater than 3.

Problem 4 (ii)

The methodology we have used is the following: for each one of the attribute names and for each one of the possible values for the attribute at hand, we have created unary predicates. For example, one of the attributes that exist in the car database is the "buying" price. One of the values from the domain of the "buying" attribute is say, "vhigh". By connecting those two symbolic names we create the unary predicate which is called "buying_vhigh", which accepts as it only argument either a car instance or a car variable. This is done for all the values of the "buying" domain, and then the above procedure is repeated for the rest 5 attributes. The predicates created this way, are shown below:

```
buying_vhigh
buying_high
buying_med
buying_low

maint_vhigh
maint_high
maint_med
maint_low

doors_two
doors_three
doors_four
doors_morethan4

persons_two
persons_four
persons_morethan4

lug_boot_small
lug_boot_med
lug_boot_big

safety_low
safety_med
safety_high
```

The next step is to somehow identify what is the value of each one of the attributes for each one of the cars. This can done easily by creating an instance of the class car for each one of the rows by attaching to the identifier "car" the row number in the file, i.e., "car23" for naming the car whose values for the attributes in the car database exist in the line with number 23. Apart from this, we have to create a type which is called "car", in order to make the various car instances, "typed" ones. This is done, by having a statement of the form "car(car23)", to denote the symbol car23 is of type "car". The last step in this preprocessing stage is to create a predicate that will indicate the target attribute, that is the"acceptability" of the car. Since this can be considered as a classification task, we choose the name "class" to denote this attribute, and we make it a binary predicate, which accepts as its first argument an object or a variable of the car type, and as the second, the specific value for the acceptability of the first argument. We should notice at this point that the same predicate should be used as a type declaration for assigning types to the values of the attributes we are trying to model. Then we can easily create the Progol file that will contain the following settings and mode declarations:

```
:- set(posonly)?
:- set(c,30)?
:- set(verbose,2)?
:- set(inflate,500)?
:- set(cover,OFF)?
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Mode Declarations
:- modeh(1,class(+car,#class))?
:- modeb(1,buying_vhigh(+car))?
:- modeb(1,buying_high(+car))?
:- modeb(1,buying_med(+car))?
:- modeb(1,buying_low(+car))?
:- modeb(1,maint_vhigh(+car))?
:- modeb(1,maint_high(+car))?
:- modeb(1,maint_med(+car))?
:- modeb(1,maint_low(+car))?
:- modeb(1,doors_two(+car))?
:- modeb(1,doors_three(+car))?
:- modeb(1,doors_four(+car))?
:- modeb(1,doors_morethan4(+car))?
:- modeb(1,persons_two(+car))?
:- modeb(1,persons_four(+car))?
:- modeb(1,persons_morethan4(+car))?
:- modeb(1,lug_boot_small(+car))?
:- modeb(1,lug_boot_med(+car))?
:- modeb(1,lug_boot_big(+car))?
:- modeb(1,safety_low(+car))?
:- modeb(1,safety_med(+car))?
:- modeb(1,safety_high(+car))?

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Type Declarations
class(unacc). class(acc). class(good). class(vgood).
```

The above Progol file segment specifies that we expect from the system to learn the "class/2" predicate, based on all the other predicates we have presented previously. As far as the training and the test set is concerned we have followed the approach to use half of the data to train the system and the rest half of them to test the system. Because of the knowledge we have about the domain, we use the following technique in order to intelligently generate the training and the test set. First of all we split the initial database into 4 files, based on the value of the target attribute (acceptability). This has as a result a file with all rows classified as 'unacc' to be created, another one with all of its rows 'acc' to be created, yet another one with 'good', and the last with the classification of 'vgood' for all of its rows. We split each one of the four generated files into two files: the first one contains the first half of the rows and the other one, the rest of the rows. Now, we put all of the four files that contain the first half of the rows into one file. We do the same for the other four files that contains the second portion. In this way, we have split the original database into two files, where both of them contain the same number of lines and also both of them contain the same number of examples for each category of the classifying attribute. We use the first file to train the Progol system to learn the "class" predicate, and the second one to test for the accuracy of the learned predicate by following the approach used in the Progol manual. We are attaching the rules learned and the contigency table created using the procedure described just previously.

```
class(A,unacc) :- maint_vhigh(A).
class(A,unacc) :- persons_two(A).
class(A,unacc) :- safety_low(A).
class(A,unacc) :- buying_vhigh(A), maint_high(A).
class(A,unacc) :- lug_boot_small(A), safety_med(A).
class(A,acc) :- buying_high(A), maint_low(A).
class(A,acc) :- buying_med(A), maint_vhigh(A).
class(A,good) :- buying_med(A), maint_low(A).
class(A,good) :- buying_low(A), maint_med(A).
class(A,vgood) :- buying_med(A), maint_med(A).
class(A,vgood) :- buying_med(A), safety_high(A).
class(A,vgood) :- buying_low(A), maint_high(A).
class(A,unacc) :- doors_two(A), persons_morethan4(A), lug_boot_small(A).
class(A,unacc) :- doors_two(A), lug_boot_med(A), safety_med(A).
class(A,acc) :- buying_high(A), maint_high(A), persons_morethan4(A).
class(A,acc) :- buying_high(A), persons_four(A), safety_high(A).
class(A,acc) :- maint_med(A), persons_morethan4(A), safety_med(A).
class(A,acc) :- maint_med(A), persons_morethan4(A), safety_high(A).
class(A,acc) :- maint_low(A), persons_four(A), safety_high(A).
class(A,acc) :- maint_low(A), persons_morethan4(A), safety_med(A).
class(A,acc) :- doors_morethan4(A), persons_four(A), lug_boot_med(A).
class(A,acc) :- persons_four(A), lug_boot_big(A), safety_med(A).
class(A,unacc) :- doors_three(A), persons_four(A), lug_boot_med(A),
```

```
        safety_med(A).
class(A,acc) :- buying_vhigh(A), maint_med(A), persons_four(A),
        safety_high(A).
class(A,acc) :- buying_vhigh(A), maint_low(A), persons_morethan4(A),
        safety_high(A).
class(A,acc) :- doors_four(A), persons_four(A), lug_boot_med(A),
        safety_med(A).
[Total number of clauses = 26]

yes
|- consult(testback)?
yes
|- test(testposi)?

[PREDICATE class/2]

Contingency table=    ________A________~A
                   P|      701|        0|       701
                    |(  701.0)|(    0.0)|
                  ~P|      164|        0|       164
                    |(  164.0)|(    0.0)|
                     ~~~~~~~~~~~~~~~~~~~
                           865         0        865
[Overall accuracy= 81.04% +/- 1.33%]
```

Problem 4 (iii)

First of all we should mention that by using the ILP, we are making use of two powerful paradigms, the inductive learning and the logic programming. The first one can be effectively used for generalizations based on a large number of examples, and the second one can be utilized for its high level expressiveness. Thus, we can say that in the first place the generated results (rules) will be highly comprehensible by the domain expert in order to evaluate or to apply them. The simplicity of the domain can give us extra accuracy without sacrifying the time efficiency in the following sense. It is known that the reason that attribute-based paradigms like rule induction and the like fail to give us a complete picture in the discovered patterns is because they cannot relate information from various tables and to extract all the required knowledge that can be related to the relation under study. But we can admit, that the domain is simple enough, and generic in nature so that the attributes used can give us generic but highly accurate results. This also can be said because the values we have for all the attributes cover the domain completely.