**Project 2**

**Using Genetic Algorithm for Associative Rule Selection**

Jafet Morales, Levent Sari

**Abstract**

**To use a genetic algorithm for finding the best rules for associative rule based classification.**

**INTRODUCTION**

In genetic algorithm based optimization, a population of solutions to a problem is evolved in order to find the best solution. The best solution is the one that minimizes a fitness function.

**Algorithm**

In a genetic algorithm at each iteration, childs solutions are born from parent solutions. Particularly in the MatLab genetic algorithm, three types of child are born: elite child, which are equal to individuals in the current generation with the best finess values, crossover children, which are created by combining the solutions of the parents, and mutation children, which are created by introducing random changes to a single parent.

We can use a genetic algorithm in order to find the best set of rules for associative rule based classification. In this case each associative rule is a gene and a vector holding all rule numbers is the genotype. Since it is better to use a small number of rules in order to prevent the final solution from being overfitted to the training data, we used only 10 rules in the genotype. An example genotype is shown below:

x=[ 130 131 2 131 129 5 132 132 130 132]

As seen in the above vector, no constrains were placed so that a single rule was used more than once. This means that the final genotype can be less than 10 rules in size, since two rules that are equal reduce to 1 when applying the rules.

In order for the genetic algorithm to reach an acceptable solution faster, certain constrains can be placed on the genotype. In this case, the genotype was required to be made of integer numbers in the range from 1 to the total number of rules in the database. The algorithm can be stopped depending on a time limit, number of generations limit, or fitness limit.

**Data Set Information:**

The examined group comprised kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected for   
the experiment. High quality visualization of the internal kernel structure was detected using a soft X-ray technique. It is non-destructive and considerably cheaper than other more sophisticated imaging techniques like scanning microscopy or laser technology.

**Results**

We ran the genetic algorithm with 133 associative rules to choose from and a genotype of size 10 and an initial population of 100 members. The algorithm was stopped when the fitness improvement was less than a threshold over a certain number of iterations. This happened at generation 51. Below is a subset of this generation.

population =

27 88 123 92 28 84 78 132 85 133

88 86 11 97 22 88 77 133 49 132

105 88 22 65 22 88 74 133 51 132

79 88 84 81 22 86 68 126 67 132

The top genotype is the one in the first row of the population subset shown above. Its fitness is an accuracy of 86.2%.

**Conclusion**

The genetic algorithm was able to reduce the select a good subset of associative rules for categorical data classification. The next step would be to explore how population size and genotype

**References**

[1]<http://en.wikipedia.org/wiki/Artificial_neural_network>

[2] <http://en.wikipedia.org/wiki/Fuzzy_logic>

[3]<http://en.wikipedia.org/wiki/Principal_component_analysis>

[4]<http://archive.ics.uci.edu/ml/machine-learning-databases/00236/>

**Appendix**

function fitness=getFitness(x,theSet,classes,ruleFeatsList,ruleCatsList,supportList,confList,classList)

vruleFeatsList=ruleFeatsList(x,:);

vruleCatsList=ruleCatsList(x,:);

vsupportList=supportList(:,x);

vconfList=confList(:,x);

vclassList=classList(:,x);

passed=[];

for ix=1:size(vruleFeatsList,1)

passed(:,ix)=applyRule(theSet,vruleFeatsList(ix,:),vruleCatsList(ix,:));

end

[gotten]=classify(vconfList,vclassList,vsupportList,passed);

accuracy=sum(gotten==classes)/length(classes);

fitness=1/accuracy;

function [passed]=applyRule(wholeSet,theFeats,theCats)

passed=ones(size(wholeSet,1),1);

i=1;

while((~isnan(theFeats(i)))&(i<=length(theFeats)))

p2=(wholeSet(:,theFeats(i))==theCats(i));

passed=passed&p2;

i=i+1;

end

% passed

% theFeats(1)

% theCats(1)

**Main Program**

clear all;

clc;

close all;

m=dlmread('seeds\_dataset.txt');

[wholeSet,classes]=prepData(m);

uniClasses=unique(classes);

ruleFeatsList=[];

ruleCatsList=[];

confList=[];

supportList=[];

classList=[];

for which=1:length(uniClasses);

c=uniClasses(which);

allCats=getAllCats(wholeSet);

currClasses=classes;

currSet=wholeSet;

currPos=getPosAll(classes,c);

while(currPos>0)

[theFeats,theCats]=getNewRule(currSet,currClasses,c,allCats);

[passed]=applyRule(wholeSet,theFeats,theCats);

support=sum(classes(passed)==c);

conf=support/sum(passed);

ruleFeatsList(size(ruleFeatsList,1)+1,:)=theFeats;

ruleCatsList(size(ruleCatsList,1)+1,:)=theCats;

supportList(length(supportList)+1)=support;

confList(length(confList)+1)=conf;

classList(length(classList)+1)=c;

[passed]=applyRule(currSet,theFeats,theCats);

[currSet,currClasses]=cleanPSatisfied(currSet,passed,currClasses,c);

currPos=getPosSatisfying(currClasses,ones(size(currClasses,1),1),c);

end

theSet=wholeSet;

passed=[];

for ix=1:size(ruleFeatsList,1)

passed(:,ix)=applyRule(theSet,ruleFeatsList(ix,:),ruleCatsList(ix,:));

end

% GENETIC ALGORITHM USAGE STARTS HERE

getFitnessWrapper= @(x)getFitness(x,theSet,classes,ruleFeatsList,ruleCatsList,supportList,confList,classList);

totRules=10;

[x,fval,exitflag,output,population,scores] = ga(getFitnessWrapper,totRules,[],[],[],[],1\*ones(totRules,1),size(ruleFeatsList,1)\*ones(totRules,1),[], ...

1:totRules); % variables 2 and 3 are integers

fval=1/fval\*100