%%-\*- text -\*-

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This is a PROMISE data set made publicly available in order to encourage

% repeatable, verifiable, refutable, and/or improvable predictive models

% of software engineering.

%

% If you publish material based on PROMISE data sets then, please

% follow the acknowledgment guidelines posted on the PROMISE repository

% web page http://promise.site.uottawa.ca/SERepository .

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. Title/Topic: JM1/software defect prediction

% 2. Sources:

%

% -- Creators: NASA, then the NASA Metrics Data Program,

% -- http://mdp.ivv.nasa.gov. Contacts: Mike Chapman,

% Galaxy Global Corporation (Robert.Chapman@ivv.nasa.gov)

% +1-304-367-8341; Pat Callis, NASA, NASA project manager

% for MDP (Patrick.E.Callis@ivv.nasa.gov) +1-304-367-8309

%

% -- Donor: Tim Menzies (tim@barmag.net)

%

% -- Date: December 2 2004

% 3. Past usage:

%

% 1. How Good is Your Blind Spot Sampling Policy?; 2003; Tim Menzies

% and Justin S. Di Stefano; 2004 IEEE Conference on High Assurance

% Software Engineering (http://menzies.us/pdf/03blind.pdf).

%

% -- Results:

%

% -- Very simple learners (ROCKY) perform as well in this domain

% as more sophisticated methods (e.g. J48, model trees, model

% trees) for predicting detects

%

% -- Many learners have very low false alarm rates.

%

% -- Probability of detection (PD) rises with effort and rarely

% rises above it.

%

% -- High PDs are associated with high PFs (probability of

% failure)

%

% -- PD, PF, effort can change significantly while accuracy

% remains essentially stable

%

% -- With two notable exceptions, detectors learned from one

% data set (e.g. KC2) have nearly they same properties when

% applied to another (e.g. PC2, KC2). Exceptions:

% -- LinesOfCode measures generate wider inter-data-set variances;

% -- Precision's inter-data-set variances vary wildly

%

% 2. "Assessing Predictors of Software Defects", T. Menzies and

% J. DiStefano and A. Orrego and R. Chapman, 2004,

% Proceedings, workshop on Predictive Software Models, Chicago,

% Available from http://menzies.us/pdf/04psm.pdf.

% -- Results:

%

% -- From JM1, Naive Bayes generated PDs of 25% with PF of 20%

%

% -- Naive Bayes out-performs J48 for defect detection

%

% -- When learning on more and more data, little improvement is

% seen after processing 300 examples.

%

% -- PDs are much higher from data collected below the sub-sub-

% system level.

%

% -- Accuracy is a surprisingly uninformative measure of success

% for a defect detector. Two detectors with the same accuracy

% can have widely varying PDs and PFs.

% 4. Relevant information:

%

% -- JM1 is written in "C" and is a real-time predictive ground system:

% Uses simulations to generate predictions

%

% -- Data comes from McCabe and Halstead features extractors of

% source code. These features were defined in the 70s in an attempt

% to objectively characterize code features that are associated with

% software quality. The nature of association is under dispute.

% Notes on McCabe and Halstead follow.

%

% -- The McCabe and Halstead measures are "module"-based where a

% "module" is the smallest unit of functionality. In C or Smalltalk,

% "modules" would be called "function" or "method" respectively.

%

% -- Defect detectors can be assessed according to the following measures:

%

% module actually has defects

% +-------------+------------+

% | no | yes |

% +-----+-------------+------------+

% classifier predicts no defects | no | a | b |

% +-----+-------------+------------+

% classifier predicts some defects | yes | c | d |

% +-----+-------------+------------+

%

% accuracy = acc = (a+d)/(a+b+c+d

% probability of detection = pd = recall = d/(b+d)

% probability of false alarm = pf = c/(a+c)

% precision = prec = d/(c+d)

% effort = amount of code selected by detector

% = (c.LOC + d.LOC)/(Total LOC)

%

% Ideally, detectors have high PDs, low PFs, and low

% effort. This ideal state rarely happens:

%

% -- PD and effort are linked. The more modules that trigger

% the detector, the higher the PD. However, effort also gets

% increases

%

% -- High PD or low PF comes at the cost of high PF or low PD

% (respectively). This linkage can be seen in a standard

% receiver operator curve (ROC). Suppose, for example, LOC> x

% is used as the detector (i.e. we assume large modules have

% more errors). LOC > x represents a family of detectors. At

% x=0, EVERY module is predicted to have errors. This detector

% has a high PD but also a high false alarm rate. At x=0, NO

% module is predicted to have errors. This detector has a low

% false alarm rate but won't detect anything at all. At 0<x<1,

% a set of detectors are generated as shown below:

%

% pd

% 1 | x x x KEY:

% | x . "." denotes the line PD=PF

% | x . "x" denotes the roc curve

% | x . for a set of detectors

% | x .

% | x .

% | x .

% |x .

% |x

% x------------------ pf

% 0 1

%

% Note that:

%

% -- The only way to make no mistakes (PF=0) is to do nothing

% (PD=0)

%

% -- The only way to catch more detects is to make more

% mistakes (increasing PD means increasing PF).

%

% -- Our detector bends towards the "sweet spot" of

% <PD=1,PF=0> but does not reach it.

%

% -- The line pf=pd on the above graph represents the "no information"

% line. If pf=pd then the detector is pretty useless. The better

% the detector, the more it rises above PF=PD towards the "sweet spot".

%

% NOTES ON MCCABE/HALSTEAD

% ========================

% McCabe argued that code with complicated pathways are more

% error-prone. His metrics therefore reflect the pathways within a

% code module.

% @Article{mccabe76,

% title = "A Complexity Measure",

% author = "T.J. McCabe",

% pages = "308--320",

% journal = "IEEE Transactions on Software Engineering",

% year = "1976",

% volume = "2",

% month = "December",

% number = "4"}

%

% Halstead argued that code that is hard to read is more likely to be

% fault prone. Halstead estimates reading complexity by counting the

% number of concepts in a module; e.g. number of unique operators.

% @Book{halstead77,

% Author = "M.H. Halstead",

% Title = "Elements of Software Science",

% Publisher = "Elsevier ",

% Year = 1977}

%

% We study these static code measures since they are useful, easy to

% use, and widely used:

%

% -- USEFUL: E.g. this data set can generate highly accurate

% predictors for defects

%

% -- EASY TO USE: Static code measures (e.g. lines of code, the

% McCabe/Halstead measures) can be automatically and cheaply

% collected.

%

% -- WIDELY USED: Many researchers use static measures to guide

% software quality predictions (see the reference list in the above

% "blind spot" paper. Verification and validation (V\&V) textbooks

% advise using static code complexity measures to decide which

% modules are worthy of manual inspections. Further, we know of

% several large government software contractors that won't review

% software modules \_unless\_ tools like McCabe predict that they are

% fault prone. Hence, defect detectors have a major economic impact

% when they may force programmers to rewrite code.

%

% Nevertheless, the merits of these metrics has been widely

% criticized. Static code measures are hardly a complete

% characterization of the internals of a function. Fenton offers an

% insightful example where the same functionality is achieved using

% different programming language constructs resulting in different

% static measurements for that module. Fenton uses this example to

% argue the uselessness of static code measures.

% @book{fenton97,

% author = "N.E. Fenton and S.L. Pfleeger",

% title = {Software metrics: a Rigorous \& Practical Approach},

% publisher = {International Thompson Press},

% year = {1997}}

%

% An alternative interpretation of Fenton's example is that static

% measures can never be a definite and certain indicator of the

% presence of a fault. Rather, defect detectors based on static

% measures are best viewed as probabilistic statements that the

% frequency of faults tends to increase in code modules that trigger

% the detector. By definition, such probabilistic statements will

% are not categorical claims with some a non-zero false alarm

% rate. The research challenge for data miners is to ensure that

% these false alarms do not cripple their learned theories.

%

% The McCabe metrics are a collection of four software metrics:

% essential complexity, cyclomatic complexity, design complexity and

% LOC, Lines of Code.

%

% -- Cyclomatic Complexity, or "v(G)", measures the number of

% "linearly independent paths". A set of paths is said to be

% linearly independent if no path in the set is a linear combination

% of any other paths in the set through a program's "flowgraph". A

% flowgraph is a directed graph where each node corresponds to a

% program statement, and each arc indicates the flow of control from

% one statement to another. "v(G)" is calculated by "v(G) = e - n + 2"

% where "G" is a program's flowgraph, "e" is the number of arcs in

% the flowgraph, and "n" is the number of nodes in the

% flowgraph. The standard McCabes rules ("v(G)">10), are used to

% identify fault-prone module.

%

% -- Essential Complexity, or "ev(G)$" is the extent to which a

% flowgraph can be "reduced" by decomposing all the subflowgraphs

% of $G$ that are "D-structured primes". Such "D-structured

% primes" are also sometimes referred to as "proper one-entry

% one-exit subflowgraphs" (for a more thorough discussion of

% D-primes, see the Fenton text referenced above). "ev(G)" is

% calculated using "ev(G)= v(G) - m" where $m$ is the number of

% subflowgraphs of "G" that are D-structured primes.

%

% -- Design Complexity, or "iv(G)", is the cyclomatic complexity of a

% module's reduced flowgraph. The flowgraph, "G", of a module is

% reduced to eliminate any complexity which does not influence the

% interrelationship between design modules. According to McCabe,

% this complexity measurement reflects the modules calling patterns

% to its immediate subordinate modules.

%

% -- Lines of code is measured according to McCabe's line counting

% conventions.

%

% The Halstead falls into three groups: the base measures, the

% derived measures, and lines of code measures.

%

% -- Base measures:

% -- mu1 = number of unique operators

% -- mu2 = number of unique operands

% -- N1 = total occurrences of operators

% -- N2 = total occurrences of operands

% -- length = N = N1 + N2

% -- vocabulary = mu = mu1 + mu2

% -- Constants set for each function:

% -- mu1' = 2 = potential operator count (just the function

% name and the "return" operator)

% -- mu2' = potential operand count. (the number

% of arguments to the module)

%

% For example, the expression "return max(w+x,x+y)" has "N1=4"

% operators "return, max, +,+)", "N2=4" operands (w,x,x,y),

% "mu1=3" unique operators (return, max,+), and "mu2=3" unique

% operands (w,x,y).

%

% -- Derived measures:

% -- P = volume = V = N \* log2(mu) (the number of mental

% comparisons needed to write

% a program of length N)

% -- V\* = volume on minimal implementation

% = (2 + mu2')\*log2(2 + mu2')

% -- L = program length = V\*/N

% -- D = difficulty = 1/L

% -- L' = 1/D

% -- I = intelligence = L'\*V'

% -- E = effort to write program = V/L

% -- T = time to write program = E/18 seconds

% 5. Number of instances: 10885

% 6. Number of attributes: 22 (5 different lines of code measure,

% 3 McCabe metrics, 4 base Halstead measures, 8 derived

% Halstead measures, a branch-count, and 1 goal field)

% 7. Attribute Information:

%

% 1. loc : numeric % McCabe's line count of code

% 2. v(g) : numeric % McCabe "cyclomatic complexity"

% 3. ev(g) : numeric % McCabe "essential complexity"

% 4. iv(g) : numeric % McCabe "design complexity"

% 5. n : numeric % Halstead total operators + operands

% 6. v : numeric % Halstead "volume"

% 7. l : numeric % Halstead "program length"

% 8. d : numeric % Halstead "difficulty"

% 9. i : numeric % Halstead "intelligence"

% 10. e : numeric % Halstead "effort"

% 11. b : numeric % Halstead

% 12. t : numeric % Halstead's time estimator

% 13. lOCode : numeric % Halstead's line count

% 14. lOComment : numeric % Halstead's count of lines of comments

% 15. lOBlank : numeric % Halstead's count of blank lines

% 16. lOCodeAndComment: numeric

% 17. uniq\_Op : numeric % unique operators

% 18. uniq\_Opnd : numeric % unique operands

% 19. total\_Op : numeric % total operators

% 20. total\_Opnd : numeric % total operands

% 21: branchCount : numeric % of the flow graph

% 22. defects : {false,true} % module has/has not one or more

% % reported defects

% 8. Missing attributes: none

% 9. Class Distribution: the class value (defects) is discrete

% false: 2106 = 19.35%

% true: 8779 = 80.65%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%