On Strategies for Improving Software Defect Prediction

Rahul Krishna, *Dept. of Electrical and Computer Engineering*
North Carolina State University, Email: rkrish11@ncsu.edu

Abstract—Programming inherently introduces defects into programs, as a result software systems can crash or fail to deliver an important functionality. It is very important to test a software throughly before it can be used. But an extensive testing can be prohibitively expensive or may take too much time to conduct This necessitates the use of automated software defect prediction tools. Although numerous machine learning algorithms are available to detect defects in software, but several factors undermine the accuracy of such algorithm. This paper uses Classification and Regression Trees (CART) and Random Forests to examines two approaches to counter the aforementioned problem. The first approach involves the use Synthetic Minority Oversampling Technique (also known as SMOTE). The second approach attempts to use a metaheursitic algorithm such as differential evolution to find the right set of parameters that can change the performance of the predictor.

Index Terms—Defect Prediction, Machine Learning, Differential Evolution, CART, Random Forest.

1 INTRODUCTION

The rest of this paper is organized as follows—Section 2 offers a small ilustration of the impact of SMOTE and tuning on the accuracy of the predictor. Section 3 highlights the underlying principles used in this paper. Section **??** presents the experimental setup followed by section **??** which presents the experimental results and discuss each one. Section **??** contains concluding remarks and finally section **??** talks about the future work.

Data	Symbol	Training	Testing	Training Samples	Defective	% Defective
Ant	ant	1.5, 1.6	1.7	644	124	19.25
Camel	cam	1.2, 1.4	1.6	1480	361	24.39
Ivy	ivy	1.1, 1.4	2.0	352	79	22.44
Jedit	jed	4.1, 4.2	4.3	679	127	18.70
Log4j	log	1.0, 1.1	1.2	244	71	29.09
Lucene	luc	2.0, 2.2	2.4	442	235	53.16
Poi	poi	2.0, 2.5	3.0	699	285	40.77
Synapse	syn	1.0, 1.1	1.2	379	76	20.05
Velocity	vel	1.4, 1.5	1.6	410	289	70.48
Xalan	xal	2.5, 2.6	2.7	1688	798	47.27

Figure 1: Attributes of the defect data sets

amc	average method complexity	e.g. number of JAVA byte codes
avg_cc	average McCabe	average McCabe's cyclomatic complexity seen in class
ca	afferent couplings	how many other classes use the specific class.
cam	cohesion amongst classes	summation of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods.
cbm	coupling between methods	total number of new/redefined methods to which all the inherited methods are coupled
cbo	coupling between objects	increased when the methods of one class access services of another.
ce	efferent couplings	how many other classes is used by the specific class.
dam	data access	ratio of the number of private (protected) attributes to the total number of attributes
dit	depth of inheritance tree	
ic	inheritance coupling	number of parent classes to which a given class is coupled (includes counts of methods and variables inherited)
lcom	lack of cohesion in methods	number of pairs of methods that do not share a reference to an instance variable.
locm3	another lack of cohesion measure	if m, a are the number of $methods, attributes$ in a class number and $\mu(a)$ is the number of methods accessing an attribute, then $lcom3 = ((\frac{1}{a}\sum_j^a \mu(a_j)) - m)/(1-m)$.
loc	lines of code	
max_cc	maximum McCabe	maximum McCabe's cyclomatic complexity seen in class
mfa	functional abstraction	number of methods inherited by a class plus number of methods accessible by member methods of the class
moa	aggregation	count of the number of data declarations (class fields) whose types are user defined classes
noc	number of children	
npm	number of public methods	
rfc	response for a class	number of methods invoked in response to a message to the object.
wmc	weighted methods per class	
defect	defect	Boolean: where defects found in post-release bug-tracking systems.

Figure 2: OO measures used in our defect data sets. Last line is the dependent attribute (whether a defect is reported to a post-release bug-tracking system).

Rank	Treatment	Med	IQR
1	RF	41.0	3.0
2	CART	44.0	3.0
3	CART (SMOTE)	70.0	2.0
4	RF (SMOTE)	78.0	1.0

(a) ant

Rank	Treatment	Med	IQR
1	RF	39.0	1.0
2	CART	43.0	2.0
3	CART (SMOTE)	56.0	2.0
4	RF (SMOTE)	60.0	2.0

(b) Camel

Rank	Treatment	Med	IQR
1	RF (SMOTE)	0.0	0.0
1	CART (SMOTE)	15.0	15.0
2	RF	50.0	1.0
3	CART	56.0	1.0

(c) Ivy

Rank	Treatment	Med	IQR
1	RF	0.0	0.0
1	CART (SMOTE)	84.0	1.0
1	RF (SMOTE)	88.0	1.0
1	CART	93.0	0.0

(d) Jedit

Rank	Treatment	Med	IQR
1	CART	36.0	3.0
1	RF	40.0	4.0
2	RF (SMOTE)	53.0	6.0
2	CART (SMOTE)	54.0	4.0

(e) POI

Rank	Treatment	Med	IQR
1	RF (SMOTE)	2.0	2.0
2	CART (SMOTE)	14.0	5.0
3	RF	22.0	2.0
4	CART	41.0	2.0

(f) Log4j

Rank	Treatment	Med	IQR
1	CART	47.0	1.0
2	RF	51.0	1.0
2	CART (SMOTE)	50.0	4.0
3	RF (SMOTE)	56.0	3.0

(g) Lucene

Rank	Treatment	Med	IQR
1	RF	51.0	0.0
1	CART	53.0	0.0
1	CART (SMOTE)	56.0	10.0
1	RF (SMOTE)	56.0	1.0

(h) PBeans

Rank	Treatment	Med	IQR
1	CART (SMOTE)	63.0	1.0
2	RF (SMOTE)	68.0	2.0
3	CART	70.0	2.0
3	RF	70.0	2.0

(i) Velocity

Rank	Treatment	Med	IQR
1	RF	24.0	1.0
2	CART	52.0	18.0
2	CART (SMOTE)	59.0	2.0
2	RF (SMOTE)	60.0	1.0

(j) Xalan

Table 1: Performance scores (g values) for the data sets.

Algorithm 1 Pesudocode for DE with Early Termination

Require: $np = 10,\ f = 0.75,\ cr = 0.3,\ life = 5,\ Goal \in \{pd, f, ...\}$
Ensure: S_{best}

```

function DE(np, f, cr, life, Goal)
    Population ← InitializePopulation(np)
    S_best ← GetBestSolution(Population)
    while life > 0 do
        NewGeneration ← ∅
        for i = 0 → np − 1 do
            S_i ← Extrapolate(Population[i], Population, cr, f)
            if Score(S_i) ≥ Score(Population[i]) then
                NewGeneration.append(S_i)
            else
                NewGeneration.append(Population[i])
            end if
        end for
        Population ← NewGeneration
        if ¬ Improve(Population) then
            life− = 1
        end if
        S_best ← GetBestSolution(Population)
    end while
    return S_best
end function
function SCORE(Candidate)
    set tuned parameters according to Candidate
    model ← TrainLearner()
    result ← TestLearner(model)
    return Goal(result)
end function
function EXTRAPOLATE(old, pop, cr, f)
    a, b, c ← threeOthers(pop, old)
    newf ← ∅
    for i = 0 → np − 1 do
        if cr < random() then
            newf.append(old[i])
        else
            if typeof(old[i]) == bool then
                newf.append(not old[i])
            else
                newf.append(trim(i,(a[i] + f ∗ (b[i] − c[i]))))
            end if
        end if
    end for
    return newf
end function
```

2 MOTIVATING EXAMPLE

3 BACKGROUND NOTES

3.1 Defect Prediction

3.2 SMOTE

3.3 The Classifiers

3.4 Differential Evolution

4 EXPERIMENTAL SETUP

4.1 Data Sets

5 EXPERIMENTAL RESULTS