Research Study Project Submittal Preparation

I. Project description and References (1 Point)

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, 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.

References

[1] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16(1), 321-357.

[2] Pelayo, L.; Dick, S., "Applying Novel Resampling Strategies To Software Defect Prediction," Fuzzy Information Processing Society, 2007. NAFIPS '07. Annual Meeting of the North American , June 2007.

[3] Fu, W., & Menzies, T. "Analytics Without Parameter Tuning Considered Harmful?", Unpublished manuscript, North Carolina State University, Raleigh NC.

[4] Menzies, Tim, et al. "Problems with precision: A response to comments on data mining static code attributes to learn defect predictors." IEEE Transactions on Software Engineering 33.9 (2007): 637.

[5] E. Barr P. Devanbu F. Rahman, S. Khatri. Comparing static bug finders and statistical prediction. ICSE14, 2014.

[6] Tim Menzies, Carter Pape, Mitch Rees-Jones, and Rahul Krishna. The promise repository of empirical software engineering data, Feb 2015.

[7] Scott, A. J., & Knott, M. (1974). A cluster analysis method for grouping means in the analysis of variance. Biometrics, 507-512.

II. Models and Equations (2 Points)

The proposed techniques do not use any explicit models or equations. I am therefore including a sample tree generated by CART (see 1) and the pseudocode of the Differential Evolution algorithm 2

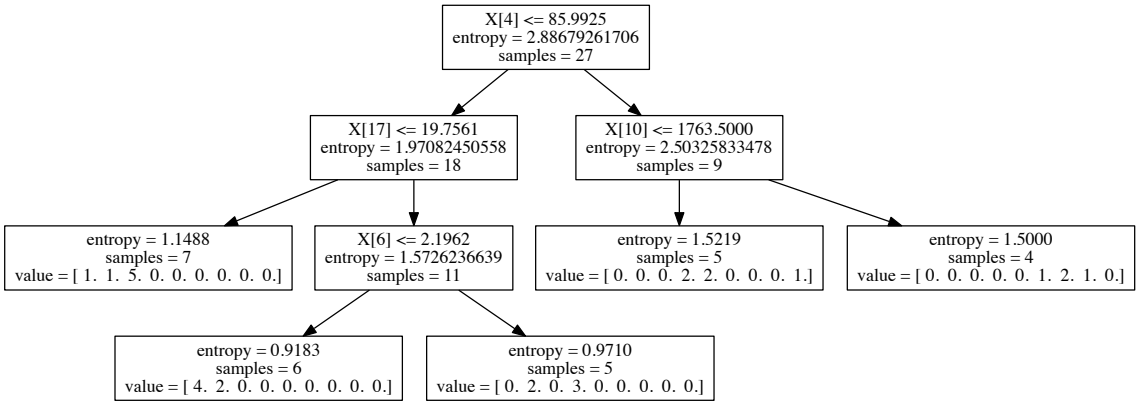

Figure 1: CART Tree example

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

```
1:
2: function DE(np, f, cr, life, Goal)
3:     Population ← InitializePopulation(np)
4:     S_best ←GetBestSolution(Population)
5:     while life > 0 do
6:         NewGeneration ← ∅
7:         for i = 0 → np − 1 do
8:             S_i ← Extrapolate(Population[i], Population, cr, f)
9:             if Score(S_i)≥Score(Population[i]) then
10:                NewGeneration.append(S_i)
11:            else
12:                NewGeneration.append(Population[i])
13:            end if
14:        end for
15:        Population ← NewGeneration
16:        if ¬ Improve(Population) then
17:            life− = 1
18:        end if
19:        S_best ← GetBestSolution(Population)
20:    end while
21:    return S_best
22: end function
23: function Score(Candidate)
24:    set tuned parameters according to Candidate
25:    model ←TrainLearner()
26:    result ←TestLearner(model)
27:    return Goal(result)
28: end function
29: function Extrapolate(old, pop, cr, f)
30:    a, b, c ← threeOthers(pop, old)
31:    newf ← ∅
32:    for i = 0 → np − 1 do
33:        if cr < random() then
34:            newf.append(old[i])
35:        else
36:            if typeof(old[i]) == bool then
37:                newf.append(not old[i])
38:            else
39:                newf.append(trim(i,(a[i] + f ∗ (b[i] − c[i]))))
40:            end if
41:        end if
42:    end for
43:    return newf
44: end function
```

Figure 2: Pesudocode for DE with Early Termination

III. METRICS, GRAPHS TABLES AND CHARTS (1 POINT)

Figure 3 lists all the metrics used by the datasets. Figure 4 contains the preliminary results of my tests.

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 3: 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	Quartiles
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	Quartiles
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	Quartiles
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	Quartiles
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	Quartiles
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	Quartiles
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	Quartiles
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	Quartiles
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	Quartiles
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	Quartiles
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

Figure 4: Scottknott rankings of performance scores (g values) for the data sets.