The Consolidated Tree Construction Algorithm in Imbalanced Defect Prediction Datasets

Igor Ibarguren[1] Jesús M. Pérez[1] Javier Mugerza [1]
Daniel Rodriguez [2] Rachel Harrison[3]

[1]University of the Basque Country, Spain

[2]University of Alcala, Spain
[3]Oxford Brookes University, UK

IEEE CEC 2017

Outline

Outline

Software Defect Prediction

- Find error prone modules in software
- Models could also be ranked

It can be used to prioritise testing, allocate resources, inspections, etc.

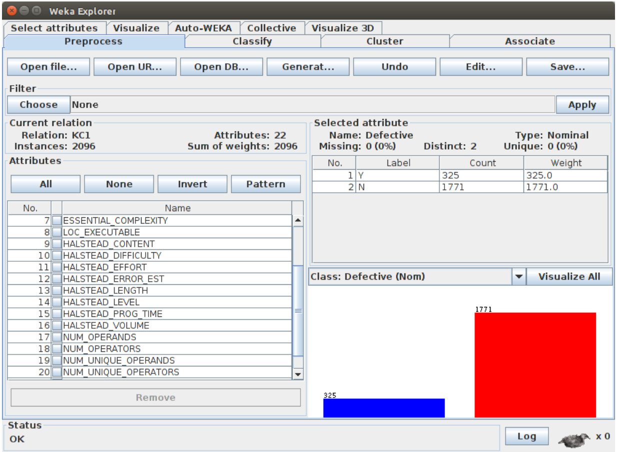
Outline

1. Introduction
   - Software Defect Prediction
   - Imbalance Data
2. Experimental Work
   - Datasets
   - Classifiers
   - Evaluation
   - Running of the Experiments
   - Results
3. Conclusions and Future Work

Imbalance data

- Most publicly available datasets in software defect prediction are highly imbalanced, i.e., samples of non-defective modules vastly outnumber the defective ones.
- Data mining algorithms generate poor models because they try to optimize the overall accuracy but perform badly in classes with very few samples (minority class which is usually the one we are interested in). This is due to the fact that most data mining algorithms assume balanced datasets.
- The imbalance problem is known to affect many machine learning algorithms such as decision tress, neural networks or support vectors machines.

Imbalanced Data



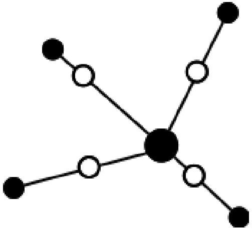
Dealing with Imbalance Data

- Sampling: Random Over-Sampling (ROS) or Random Under-Sampling (RUS) are based on adding or removing instances of the training dataset.
- Cost-Sensitive Classifiers (CSC) penalises differently the type of errors
- Ensembles: Bagging (Bootstrap aggregating), boosting and stacking (Stacked generalization) which combines different types of models
- Robust algorithms: algorithms designed to work with unbalanced data

Over-Sampling: SMOTE

In addition to ROS, there are more *intelligent* approaches to generate synthetic data points.

- SMOTE over-sampling approach in which the minority class is oversampled by creating synthetic instances along the line segments joining any/all of the *k* minority class nearest neighbors (NN).



Cost-Sensitive Classifiers (CSC)

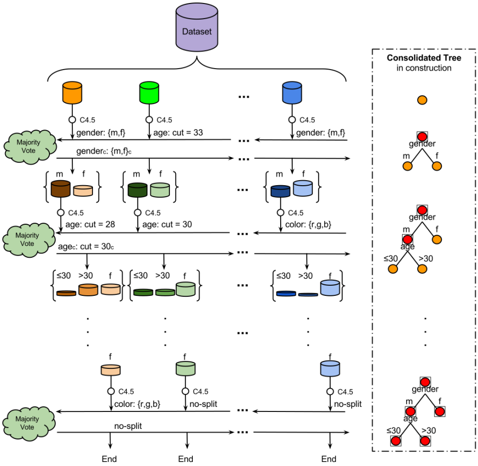
- The idea is to penalise differently the different types of error (in binary classification, the false positives and false negatives).
- Adapt classifiers to handle imbalanced datasets by either
  - adding weights to instances (if the base classifier algorithm allows this) or resampling the training data according to the costs assigned to each class in a predefined cost matrix
  - generating a model that minimises the expected cost

Whitebox vs Blackbox algorithms

- Decision trees and rules generate rules capable of explaining why decisions are made
- This is in opposition black-block approaches such as neural-networks or meta-learners that cannot explain why a selection was done.

In this work we analyse white-box approaches with an algorithm that considers imbalanced data while it is being created, J48 Consolidated

J48 Consolidated



Outline

Datasets

We have used available software defect prediction datasets generated from projects carried out at NASA.
These datasets are available in two different versions from:

- The Tera-PROMISE repository:
  `http://openscience.us/repo/`
- And the original one which has curated by Shepperd et al. who analysed different problems and differences with these datasets and curated the repository.

Datasets
Attribute Definition

	Metric	*Definition*
McCabe	*LoC*	McCabe's Lines of code
	$v(g)$ $ev(g)$ $iv(g)$	Cyclomatic complexity Essential complexity Design complexity
Halstead base	*uniqOp* *uniqOpnd* *totalOp* *totalOpnd*	Unique operators, n_1 Unique operands, n_2 Total operators, N_1 Total operands, N_2
Halstead Derived	n L V d i e b t lOCode lOComment lOBlank lOCodeAndComment	Vocabulary, $n = n_1 + n_2$ Program length, $N = N_1 + N_2$ Volume, $V = N \cdot log_2(n)$ Difficulty $D = 1/L$ Intelligence Effort $e = V/L$ Error Estimate Time $T = E/18$ seconds Line count of statement Count of lines of comments Count of blank lines Count of lines of code and comments
Branch	*branchCount*	No. branches of the flow graph
Class	true, false	Reported defects?

Dataset
Description

Table: MDP NASA Datasets Description

	# Instances D'	%Imbalance Ratio	# Attributes
CM1	344	12.21	41
JM1	9,593	18.34	22
KC1	2,095	15.51	22
KC3	200	18	41
MC1	8,737	0.78	40
MC2	127	34.65	41
MW1	264	10.23	41
PC1	759	8.04	41
PC2	1,493	1.07	41
PC3	1,125	12.44	41
PC4	1,399	12.72	41
PC5	16,962	2.96	40

Outline

Classifiers

- C4.5 (called J48 in Weka) is a decision tree where the leaves of the tree correspond to classes, nodes correspond to features, and branches to their associated values
- JRip (RIPPER) Rule algorithm: Repeated Incremental Pruning to Produce Error Reduction
- PART - Builds a partial C4.5 decision tree in each iteration and best leaf is rurned into a rule
- CART - Classification and Regression Trees Breiman et al (1984)
- J48Consolidated

Outline

Binary classifiers Evaluation

Confusion matrix

		Actual *Pos*	*Actual* *Neg*	
Pred	*Pos*	True Positive (TP)	False Positive (FP) Type I error (False alarm)	*Positive Predictive Value* (PPV)= *Confidence* $=$ *Precision* $= \frac{TP}{TP+FP}$
Pred	*Neg*	False Negative (FN) Type II error	True Negative (TN)	*Negative Predictive Value* (NPV)$= \frac{TN}{FN+TN}$
		Recall $=$ *Sensitivity* $= TP_r = \frac{TP}{TP+FN}$	*Specificity* $= TN_r = \frac{TN}{FP+TN}$	

Evaluation measures

Common used measures with imbalance data include ROC (AUC), MCC, and the $f-measure$, which are defined as:

- Area Under the ROC Curve
  $AUC = \frac{1+TP_r-FP_r}{2}$
- Matthews Correlation Coefficient (MCC)
  $MCC = \frac{TP\times TN-FP\times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$

Outline

Running of the Experiments

- All algorithms were run using the WEKA environment, the Experimenter tool.
- Results were obtained with 5 runs, each run is a 5-fold CV, i.e., 5x5CV.
- The *t*-test was used to compare with the base classifier provided by WEKA as well as the aligned Friedman ranking.

Outline

D’ Results using AUC

	J48CTC	J48	J48+SMOTE	J48+Cost	JRIP	PART	CART
CM1	.67	.56	.59	.64	.52 •	.63	.50 •
JM1	.67	.67	.66	.66	.56 •	.70 ◦	.62
KC1	.74	.67	.69 •	.66 •	.59 •	.75	.68
KC3	.66	.59	.65	.67	.61	.62	.54
MC1	.89	.77 •	.81	.81	.65 •	.79	.79 •
MC2	.63	.62	.61	.58	.59	.62	.59
MW1	.61	.58	.59	.63	.58	.62	.51
PC1	.76	.70	.68	.68	.57 •	.73	.53 •
PC2	.86	.52 •	.56 •	.56 •	.50 •	.65	.50 •
PC3	.73	.65 •	.64	.68	.53 •	.71	.50 •
PC4	.84	.77	.75 •	.81	.71 •	.82	.86
PC5	.94	.77 •	.79 •	.67 •	.75 •	.91	.85 •
Avg	.75	.65	.67	.67	.60	.71	.62

◦, • statistically significant improvement or degradation

D' Results using MCC

	J48CTC	J48	J48+SMOTE	J48+Cost	JRIP	PART	CART
CM1	.24	.10	.17	.23	.05	.09	.00 •
JM1	.27	.23 •	.24	.24	.20 •	.17 •	.17 •
KC1	.34	.28	.31	.32	.26	.24 •	.21 •
KC3	.25	.22	.29	.24	.26	.23	.11
MC1	.19	.44 ○	.43 ○	.44 ○	.42 ○	.44 ○	.44 ○
MC2	.25	.21	.20	.16	.23	.26	.22
MW1	.14	.32	.15	.20	.19	.28	.00
PC1	.28	.24	.26	.30	.25	.23	.08 •
PC2	.18	.00 •	.09	.09	.01 •	.07	.00 •
PC3	.33	.24	.22	.29	.10 •	.14 •	.00 •
PC4	.51	.51	.52	.51	.44	.46	.40 •
PC5	.49	.50	.54 ○	.52	.51	.42	.44
Avg	.29	.27	.29	.29	.24	.25	.17

○, • statistically significant improvement or degradation

MCC Average Rankings

Table: MCC Average Rankings of the algorithms (Aligned Friedman) and adjusted *p*-value (Holm test)

Algorithm	Ranking	p_{Holm}
J48Consolidated	27.25	—
J48Cost	27.7083	1
J48Smote	30.25	1
J48	40.5	0.5881
PART	48.8333	0.1327
JRIP	53.0833	0.05286
CART	69.5	0.0001

AUC Average Rankings

Table: AUC Average Rankings of the algorithms (Aligned Friedman) and adjusted p-value (Holm test)

Algorithm	Ranking	p_{Holm}
J48Consolidated	13.7083	—
PART	21.625	0.4266
J48Cost	39.2083	0.0208
J48Smote	44.4167	0.0061
J48	50.3333	0.0009
CART	58.4167	0
JRIP	69.7917	0

Conclusions

Conclusions

- There are some *questions* about the quality of the data
- Duplicates, noise

Future Work

- Use other datasets and better statistical tests
- Analyse duplicates and noise (meta-learning)
- Combine it with other technique such us noise filtering and feature selection