Machine Learning for Software Engineering: Case Studies in Software Reuse

Justin S. Di Stefano, Tim Menzies
Lane Department of Computer Science, West Virginia University
PO Box 6109, Morgantown, WV, 26506-6109, USA
justin@lostportal.net, tim@menzies.com

Abstract

There are many machine learning algorithms currently available. In the 21st century, the problem no longer lies in writing the learner, but in choosing which learners to run on a given data set. In this paper, we argue that the final choice of learners should not be exclusive; in fact, there are distinct advantages in running data sets through multiple learners.

To illustrate our point, we perform a case study on a reuse data set using three different styles of learners: association rule, decision tree induction, *and* treatment. *Software reuse is a topic of avid debate in the professional and academic arena; it has proven that it can be both a blessing and a curse. Although there is much debate over where and when reuse should be instituted into a project, our learners found some procedures which should significantly improve the odds of a reuse program succeeding.*

KEYWORDS: *AI algorithms, AI in software engineering, AI in data mining, machine learning, reuse, empirical studies, treatment learning, association rule learning, decision tree learning, C4.5, J4.8, J4.8 PART, APRIORI, TAR2.*

Attribute	Morisio et.al.	this paper
state:		
Application Domain	Not analyzed	×
Size of Baseline	Not analyzed	✓
Production Type	✓	×
high-level control:		
Top Management Commitment	✓	×
low-level control:		
Reuse Approach	×	✓
Domain Analysis	×	✓

Figure 1. Conclusions where we disagree with Moriso et.al. ×/✓= no/some evidence (respectively) in this data set that this attribute is relevant to determining success or failure of a reuse project.

1 Introduction

In the 21st century, there are too many machine learning tools. Users are overwhelmed with possibilities, and choosing which learners to apply to their data can be a daunting task. The challenge now is not in building or designing a learner, but in sorting through the reams of mature and efficient ones in order to find a subset which will prove useful.

A problem just as prolific and important as choosing which learners to run is making sure that you have run enough to produce a stable and viable conclusion. When faced with the task of choosing a proper subset of learners, our belief is that researchers stop to soon, i.e. they derive their conclusions after running just one learner. However, with the relative ease of running multiple learners, there is no reason why those results cannot be cross-checked with other tools. In fact, running multiple learning algorithms on a data set can produce quite varied and useful results.

To illustrate this point, we use the data set from Morisio *et.al.*'s "Success and Failure Factors in Software Engineering" [18]. This data is a collection of interview results with industrial projects involved in the introduction of reuse. The interviews were performed by Morisio, Ezran, and Tully [18]. The industrial projects presented are a subset of 288 Process Improvement Experiments funded by the European Commission. In each case, the project was chosen because it represented a serious attempt at implementing a reuse process.

In their paper, the researchers (Morisio, *et.al.*) ran the CART [5] machine learner on their data set (a list of success and failure cases for the implementation of a software

[1]Submitted to the 14th IEEE International Conference on Tools with Artificial Intelligence, November 4-6, 2002 Washington D.C. http://www.nvc.cs.vt.edu/ictai2002/. August 25, 2002. Wp ref: http://www.csee.wvu.edu/~justin/wp/02/toolsai/swreuse.tex

reuse program in various corporations), and drew their conclusions based on it's output. However, we show here that if further learning (using other learning algorithms) is performed on the data set, some different (and essential) results can be produced. Figure 1 highlights the differences between our results and those obtained by Morisio *et.al*. For a full listing of all the attributes collected, please refer to the original article [18].

In reevaluating the data from the above mentioned paper, the J4.8 [13], J4.8 PART [13], APRIORI [2], and TAR2.2 [22] machine learners were run on the data set. J4.8 is a *decision tree induction learner* (based on C4.5), which operates by splitting data into subsets for each particular value of an attribute. J4.8 PART finds *classification rules* based on partial decision trees (specifically, the decision trees produced by J4.8). APRIORI is an *association rule learner*, in that it attempts to find associations between various attributes of the data set. Finally, TAR2.2 is a *treatment learner*. A treatment learner uses an algorithm which, instead of finding classifications or associations, attempts to find one, or a conjunction of, attribute ranges which predict for an increased frequency of the best class, and/or a decreased frequency of the worst [22]. This is especially useful in cases where a best and worse class can easily be defined, such as with the success and failure of a software reuse program.

The rest of this paper shows how we generated Figure 1. We will show that using a variety of learners on a data set (instead of limiting oneself to a single class of machine learner) can produce more definitive and useful results. We will also present some conclusions about the various factors which can effect the potential success (or failure) of a software reuse program.

2 Background

2.1 Learning

The goal of learning is to find important patterns in data sets. Analyzing these data sets by hand is problematic at best, and can take substantial time and effort. It is both quicker and easier if a computer can be "taught" to search for these patterns. In order to facilitate this, many different types of learners have evolved, including (but not limited to) *decision tree learners*, *classification rule learners*, *association rule learners*, and *treatment learners*. Each of these machine learners was used in our analysis of the data set; they are described in the following sections.

2.1.1 Decision Tree Learners and J4.8

Decision Tree Learners attempt to find paths which arrive at a specific class instance. J4.8, specifically, uses a method called decision tree induction. In this method, data is split using a standard recursive splitting technique, which produces a decision tree whose leaf nodes contain training examples of one class [12]. This means that the output of the J4.8 algorithm is a "decision tree", garnered from the provided data, which suggests a path that can be taken in order to arrive at a specific class instance, i.e. success. J4.8 is a java port of the C4.5 algorithm [21].

C4.5 uses a heuristic *entropy* measure of information content to build its trees. The attribute that offers the largest *information gain* is selected as the root of a decision tree. The example set is then divided up according to which examples do/do not satisfy the test in the root. For each divided example set, the process is then repeated recursively.

The information gain of each attribute is calculated as follows. A tree C contains p examples of some class and n examples of other classes. The *information required* for the tree C is as follows:

$$I(p,n) = -\left(\frac{p}{p+n}\right) log_2 \left(\frac{p}{p+n}\right) - \left(\frac{n}{p+n}\right) log_2 \left(\frac{n}{p+n}\right)$$

Say that some attribute A has values $A_1, A_2, ...A_v$. If we select A_i as the root of a new sub-tree within C, this will add a sub-tree C_i containing those objects in C that have A_i. We can then define the expected value of the information required for that tree as the weighted average:

$$E(A) = \sum_{i=1}^{v} \left(\frac{p_i + n_i}{p+n}\right) I(p_i, n_i)$$

The information gain of branching on A is therefore:

$$gain(A) = I(p,n) - E(A)$$

2.1.2 Classification Rule Learners and J4.8 PART

A classification rule learner operates by identifying a rule that covers instances in a specific class (and excludes ones not in the class), separates them out, and continues learning on the remaining instances [13]. More specifically, J4.8 PART is a *partial decision tree rule* learner. The algorithm parses the pruned decision trees output by the J4.8 algorithm, and searches for rules which can be deduced from the tree. In some cases, classification rules can be significantly more compact and easier to read than decision trees. In addition, rules are often preferred to decision trees since each rule seems to represent an independent "nugget" of knowledge [13].

2.1.3 Association Learners and APRIORI

Association rule learners are a generalization of classification rule learners; they can predict for any attribute of the data set, not just the class [13]. Association rule learners

can find significant inter-relationships between attributes. For example, APRIORI often reveals that some attributes predict for others. In these cases, it maybe useful to remove the dependent attribute(s), since they are unnecessary during evaluation, and tend to slow down and confuse the learning process.

2.1.4 Treatment Learners and TAR2

Treatment learners differ from most other learners in that instead of attempting to find a predictor for a class, they attempt to find a *treatment* which predicts for an increased frequency of the best class and a decreased frequency of the worst. This type of learner is extremely useful when you are evaluating a data set which has clearly ordered classes (i.e., success and failure).

TAR2 seeks attribute ranges that occur more frequently in the highly scored classes than in the lower scored ones[22], and produces various rules based on those attributes. It also allows for the selection of how many attributes should be used to produce a rule. This allows for the production of a varied and extensive set of rules which can involve only a single attribute range, or a range for every attribute present in the data set. A detailed description of the mining algorithm for TAR2 follows.

TAR2 seeks attribute ranges that occur more frequently in the highly scored classes than in the lower scored classes. Let $a.r.$ be some attribute range, e.g. $DomainAnalysis = yes$, then $\Delta_{a.r.}$ is a heuristic measure of the worth of $a.r.$ to improve the frequency of the *best* class. $\Delta_{a.r.}$ uses the following definitions:

$X(a.r.)$: is the number of occurrences of that attribute range in class X; e.g. $success(DomainAnalysis.yes) = 9$.

$all(a.r.)$: is the total number of occurrences of that attribute range in all classes; e.g. $all(DomainAnalysis.yes) = 9$

$best$: the highest scoring class; e.g. $best = success$.

$rest$: the non-best class; e.g. $rest = failure$.

$score$: the score of a class X is $X.

Using these definitions, $\Delta_{a.r.}$ is calculated as follows:

$$\Delta_{a.r.} = \frac{\sum_{X \in rest}(\$best - \$X) * (best(a.r.) - X(a.r.))}{all(a.r.)}$$

A *treatment* is a subset of the attribute ranges with an outstanding $\Delta_{a.r.}$ value. To *apply* a treatment, TAR2 rejects all example entries that contradict the conjunction of the attribute ranges in the treatment. The ratio of the classes in the remaining examples is compared to the ratio of classes in the original example set. The *best treatment* is the one that most increases the relative percentage of preferred classes [22].

2.1.5 Comparison

It is difficult to find a paper, article, or study which compares multiple learning algorithms. Many people have written or spoke about the various benefits and problems with different learners, but normally those studies refer only to learners in the same class (such as classification learners). However, there are exceptions.

Almeida and Lounis did a study on the use of machine learned models for estimating correction costs [16]. In their article, they compare the NewID [4], CN2 [6], C4.5 [21] and FOIL [20] machine learning algorithms. They conclude that

> The results show that the inductive logic programming algorithms [FOIL], are superior to the top-down induction decision tree, top-down induction attribute value rules, and covering algorithms ...

While their conclusion is born out by their data and testing techniques, they make no mention of the fact that while the FOIL learner may have performed better in their test, it would behoove the practicing researcher to try *all* the available learners before attempting to reach a conclusion about their data set.

3 The Tests

In the following sections, we provide the breakdown of our testing methods and results, and those from Morisio *et.al.*'s study.

3.1 Morisio et.al.'s Test

In their study, Morisio *et.al.* chose to separate the attributes from the data set into three classes, namely *State Variables*, *High-Level Control Variables*, and *Low-Level Control Variables*. They define the different classes in this way:

- State Variables - Attributes over which a company has no control.
- High-Level Control Variables - Key high-level management decisions about a reuse program.
- Low-Level Control Variables - Specific approaches to the implementation of reuse.

Furthermore, Morisio *et.al.* chose to restrict their analysis to the state and high-level control variables. Their reasoning is that the high-level control variables temporally and/or logically precede the low-level control variables. For instance, they say

> ...a decision to introduce reuse processes both logically and temporally precedes a decision

about whether to perform domain analysis and about when to to develop assets.

While the reasoning for breaking up the attributes is sound, Morisio *et.al.*'s conclusions are skewed by their choice to exclude the low-level control variables. One of their main conclusions was that *not* addressing two or more high-level control variables led to failure [18]. Having said this, however, they say *nothing* about *which* specific approach to take in the implementation of reuse (for example, which reuse-specific processes to implement). In essence, their conclusions tell a company where to go without giving them the directions on how to get there.

When analyzing a data set, it is important that *all* the attributes be properly analyzed. Otherwise, important patterns and information may be missed. This was the problem with Morisio *et.al.*'s analysis of the data set. By choosing to exclude certain attributes, they have inappropriately constrained their learner and limited the scope of their conclusions.

3.2 Our Tests

In analyzing this data set, we have left all the variables intact and unclassified (with the sole exception of the *Repository* attribute, which was excluded when running APRIORI because it has value "*yes*" for all cases). In addition, we have run a selection of various machine learners in order to ensure that all available patterns are discovered.

For each machine learner, we compare our results with those from the original paper (by Morisio *et.al.*) [18]. The learners are presented in the order in which they were run.

In order to validate the results from the learners, a common practice called 10-way cross validation was performed. In this scheme, the data set is split into 10 separate pieces, each with the same distribution of classes. The algorithms then learn on 9 of the sets, and test the learned model on the 10th. If the error rate during cross validation is low (the exact definition of *low* can vary, depending on the domain and data set your are working with), then the learned model can be considered valid.

3.2.1 APRIORI

APRIORI was run first, since it is useful to know about any dependencies in a data set before running other learning algorithms. Results from APRIORI must be evaluated carefully, however, since some dependencies will appear which have no "real-world" meaning. What we are looking for from the output of APRIORI is a dependency which can be linked to a real-world situation, such as $TopManagementCommitment = no ==> RewardsPolicy = no$. In this data set, however, no such conclusions were found. Since APRIORI failed to generate a usable dependency, it neither confirmed nor denied the conclusions of Morisio *et.al*.

3.2.2 J4.8

In this test, the data was run through the J4.8 machine learner without any modifications. The class was simply the success or failure of the software reuse program. The resulting tree used only one attribute, *Human Factors*. The tree is shown below.

```
Human Factors = yes: success (16.0/1.0)
Human Factors = no: failure (8.0)
```

This tree has an error rate, in 10 way cross-validation, of just 4.2%.

After running the above test, we removed the attribute Human Factors from the data, and ran J4.8 again. The resulting tree is show below.

```
Reuse Processes Introduced = yes: success (15.0/1.0)
Reuse Processes Introduced = no: failure (8.0/1.0)
Reuse Processes Introduced = NA: failure (1.0)
```

The error rate on this decision tree, after running 10 way cross-validation, is 20.8%. As you can see, this tree also only uses one attribute, *Reuse Processes Introduced*.

The original evaluation performed by Morisio *et.al.* used a machine learner called CART. CART is very similar to J4.8, and also produces a decision tree. That decision tree is shown below.

```
Human Factors = yes:
   Type of software production = product-family: success
   Type of software production = isolated: failure
Human Factors = no: failure
```

The difference between the two algorithms is evident in their outputs. J4.8 is driven to find the simplest tree possible with an acceptable failure rate, while CART is attempting to find the tree with the lowest failure rate, regardless of simplicity. In this case, the CART algorithm was successful in finding a tree with a 0% failure rate by using two attributes.

3.2.3 J4.8 PART

Rule learners like J4.8 PART can simplify complicated decision trees. In this study, the decision trees from §3.2.2 are very simple. Therefore, the output from J4.8 PART is nothing more than a simple restatement of those trees. For completeness' sake, the output from the J4.8 PART algorithm is shown below for both of the decision trees from §3.2.2.

```
FIRST TREE:
Human Factors = yes: success (16.0/1.0)
: failure (8.0)
```

```
SECOND TREE:
Reuse Processes Introduced = yes: success (15.0/1.0)
: failure (9.0/1.0)
```

Once again, these results closely agree with the tree produced by CART.

3.2.4 TAR2

As was discussed above, TAR2 is a "Treatment Learner", which can produce very different results than normal classification and decision tree learners. Running this learner generated 3 stable (under 10 way cross-validation) and useful results. As was explained above, the results from TAR2 are attribute ranges which select for the best class (in this case, success). The 3 attribute ranges which are most useful in selecting for success of a reuse project are:

```
Size of Baseline = L
Domain Analysis = yes
Reuse Approach = tight
```

None of these attributes appear in the original evaluation performed by Morisio *et.al*. This is a combination of the choice by Morisio *et.al.* to leave out the low-level control variables, and a result of the limiting depth of running only a single learner.

4 Conclusions

Our conclusions take 2 forms:

- About reuse
- About learning

§4.1 will present a general decision path to use when instituting a reuse program, and §4.2 will present our view on the use of multiple machine learning algorithms.

4.1 Software Reuse

After applying multiple machine learners, several conclusions can be drawn about the potential success of a reuse program. First and foremost, and in agreement with Morisio *et.al.*'s conclusions, the single largest deciding factor for success is "Human Factors". This conclusion makes sense, as people are more likely to be successful in implementing reuse if they are properly trained and aided in doing so.

Next, and in no particular order, the additional factors effecting reuse are "Reuse Processes Introduced", "Size of Baseline", "Reuse Approach", and "Domain Analysis". To summarize the basic steps that should be taken in order to increase the chance of success for a software reuse program, we present a small rule table in two parts. The first part is areas where we agree with the conclusions by Morisio *et.al.*, and the second is where our conclusions differ from theirs.

IN AGREEMENT:
1. Train, aid, and entice people to write reusable code.
2. Introduce reuse specific processes.

DIFFERING:
3. Don't bother trying to get reusable code out of small projects.
4. Perform Domain Analysis.
5. Make sure that reusable work products are tightly coupled.

These steps should aid in beginning a reuse program in any company. In fact, based on the Morisio *et.al.* data, our learners are telling us that these 5 simple steps will increase the chances for success dramatically.

4.2 Multiple Learners

As we have shown, a single learner (or single type of leaner) is not sufficient to find all the applicable patterns buried in a data set. With the increasing speed and memory capacity of computers, it is no longer a matter of days to run an efficient machine learner; in many cases, it is not even a matter of hours. Given the relative ease of running multiple learners, our thesis is that a good analysis of a data set *should* use multiple learners.

Many mature machine learning tools are freely available through public domain downloads. For example, the J4.8, J4.8 PART, and APRIORI implementations used in this paper came from the WEKA [13] toolkit `http://www.cs.waikato.ac.nz/ml/weka/`. Also, our TAR2 machine learner is available from `http://www.ece.ubc.ca/twiki/bin/view/Softeng/TreatmentLearner`.

Acknowledgements

This research was conducted at West Virginia University under NASA contract NCC2-0979. The work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA IV&V Facility. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not constitute nor imply its endorsement by the United States Government.

References

[1] C. Abts, B. Clark, S. Devnani-Chulani, E. Horowitz, R. Madachy, D. Reifer, R. Selby, and B. Steece. COCOMO II

model definition manual. Technical report, Center for Software Engineering, USC,, 1998. http://sunset.usc.edu/COCOMOII/cocomox.html#downloads.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases*, 1994. Available from http://www.almaden.ibm.com/cs/people/ragrawal/papers/vldb94_rj.ps.

[3] K.-D. Althoff, M. Nick, and C. Tautz. Improving organizational memories through user feedback. In F. Bomarius, editor, *Proc. of the Workshop on Learning Software Organizations (LSO) (in conjunction with the11th International Conference on Software Engineering and Knowledge Engineering, SEKE'99, Kaiserslauten, Germany)*, pages 27–44, June 1999.

[4] R. Boswell. Manual for newid, January 1990.

[5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. Technical report, Wadsworth International, Monterey, CA, 1984.

[6] P. Clark and T. Ng. The cn2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[7] P. Coad, D. North, and M. Mayfield. *Object Models: Strategies, Patterns, and Applications*. Prentice Hall, 1997.

[8] P. Cohen, V. Chaudhri, A. Pease, and R. Schrag. Does prior knowledge facilitate the development of knowledge-based systems? In *AAAI'99*, 1999.

[9] W. Cunningham. The checks pattern language of information integrity. In J. Coplien and D. Schmidt, editors, *Pattern Languages of Program Design*. Addison-Wesley, 1995. Also avialable at *http://c2.com/ppr/checks.html*.

[10] W. Frakes and C. Fox. Sixteen questions about software reuse. *Communications of the ACM*, 38(6):75–87, June 1995.

[11] E. Guerrieri. Reuse success - when and how? In *Proceedings of WISR9: The 9th annual workshop on Institutionalizing Software Reuse*, 1999. Available from http://www/umcs.maine.edu/~ftp/wisr/wisr9/final-papers/Guerrieri.html.

[12] L. Holder. Intermediate decision trees, 1995. Available from http://www-cse.uta.edu/~holder/pubs/ijcai95.ps.

[13] I.H.Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Academic Press, 2000.

[14] N. Kerth. Caterpillar's fate: A pattern language for transformation from analysis to design. In J. Coplien and D. Schmidt, editors, *Pattern Languages of Program Design*. Addison-Wesley, 1995. Also available from *http://c2.com/ppr/catsfate.html*.

[15] J. Lee, M. Gruninger, Y. Jin, T. Malone, A. Tate, G. Yost, and other members of the PIF working group. The PIF Process Interchange Format and framework. *Knowledge Engineering Review*, 13(1):91–120, Feb. 1998.

[16] H. L. Mauricio A. de Almeida and W. L. Melo. An investigation on the use of machine learned models for estimating correction costs. In *Proceedings of the 21st International Conference on Software Engineerin. Kyoto, Japan*, 1997.

[17] T. Menzies. Se/ke reuse research: Common themes and empirical results. In S. Chang, editor, *Handbook of Software Engineering and Knowledge Engineering, Volume II*. World-Scientific, 2002. Available from http://tim.menzies.com/pdf/00reuse.pdf.

[18] M. Morisio, M. Ezran, and C. Tully. Success and failure factors in software reuse. *IEEE Transactions on Software Engineering*, 28(4):340–357, 2002.

[19] D. Perry. Some holes in the emperor's reused clothes. In *Proceedings of WISR9: The 9th annual workshop on Institutionalizing Software Reuse*, 1999. Available from http://www/umcs.maine.edu/~ftp/wisr/wisr9/final-papers/Perry.html.

[20] J. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, August 1990.

[21] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992. ISBN: 1558602380.

[22] T.Menzies and Y. Hu. The tar2 treatment learner, 2002. Available from http://www.ece.ubc.ca/twiki/pub/Softeng/TreatmentLearner/intro.pdf.

[23] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. *The Knowledge Engineering Review*, 13(1), Feb. 1998.