The Inductive Software Engineering Manifesto: Principles for Industrial Data Mining

Tim Menzies
Lane Dept. of CS & EE,
West Virginia University
Morgantown, USA
tim@menzies.us

Christian Bird,
Tom Zimmermann,
Wolfram Schulte
Microsoft Research, Redmond, USA
{*cabird|tzimmer|schulte*}*@microsoft.com*

Ekrem Kocaganeli
Lane Dept of CS & EE,
West Virginia University
Morgantown, USA
kocaguneli@gmail.com

***Abstract*—The practices of industrial and academic data mining are very different. These differences have significant implications for (a) how we manage industrial data mining projects; (b) the direction of academic studies in data mining; and (c) training programs for engineers who seek to use data miners in an industrial setting.**

***Keywords*-data mining, induction, process**

I. Introduction

It is important for industrial practitioners to document their methods. Such documentation lets newcomers become more effective, sooner. Accordingly, this article documents our *principles of industrial inductive software engineering*.

Engineering is the process of developing repeatable processes that generate products to an acceptable standard within resource constraints (e.g. time). Hence, inductive software engineering is that branch of SE focusing on the delivery of data mining-based software applications to users. In our view, the practices of industrial inductive engineering are (a) significantly different to those of academic data mining research; and that (b) those differences have never really been clearly articulated. Hence, this article.

This paper is a result of a reflection on our applied data mining work (e.g. for defect prediction [29], [31], social metrics [6], effort estimation [25], test case generation [2], and others [3]–[5], [22], [39], [40]). In an initial informal stage, we maintained the whiteboard of Figure 2. Any visitor would be asked "what characterizes the difference between academic and industrial data mining?". The results of that informal analysis was then formalized and systematized into the seven *principles* of Figure 1 and a dozen other *tips*.

This paper is structured as follows. We state our guiding principle: *users before algorithms*. This user-focus will lead to three extensions to traditional descriptions of applied data mining: user involvement, cycle evolution, and early feedback. After that, we discuss some details of industrial inductive engineering. We conclude with some notes on the implications of our work for project management, training, and research.

Inductive Software Engineering Manifesto

Users before algorithms
Plan for scale
Early feedback
Be open-minded
Do smart learning
Live with the data you have
Broad skill set, big toolkit

Figure 1. Manifesto, version 1.0.

Where possible, the arguments of this paper will be based in terms of standard academic rhetoric. However, caveat emptor, much of this paper draws from personnel experience and cannot be "proved" in some formal sense. As argued at a recent panel on the relationship between industry and academia at ICSE 2011 (see http://goo.gl/YHsYY), it may be inappropriate to ask for such proofs on industrial perspectives. Like Aranda et al. [1], we believe it useful to:

> *... push for a better dissemination of our results and methods, making the argument that there is more to science than trial runs and statistical significance, and helping practitioners distinguish between good and bad science, whatever its methods of choice.* [1]

II. User-Focused Development

The main difference we see between academic data mining research and our industrial practices is that the former is focused on algorithms and the latter is focused on *users*: By this term, we do not mean the end-user of a product. Rather, we mean the community providing the data and domain insights vital to a successful project. Users provide the funding for the project and, typically, need to see a value added benefit, very early in a project. Hence:

Submitted to MALETS 2011: the *International Workshop on Machine Learning Technologies in Software Engineering*, Lawrence, Kansas, U.S.A. November 12, 2011. In association with the 26th International Conference on Automated Software Engineering.

See http://bit.ly/o02QZJ for an earlier draft of this paper.

This work was partially funded by the Qatar/West Virginia University research grant NPRP 09-12-5-2-470.

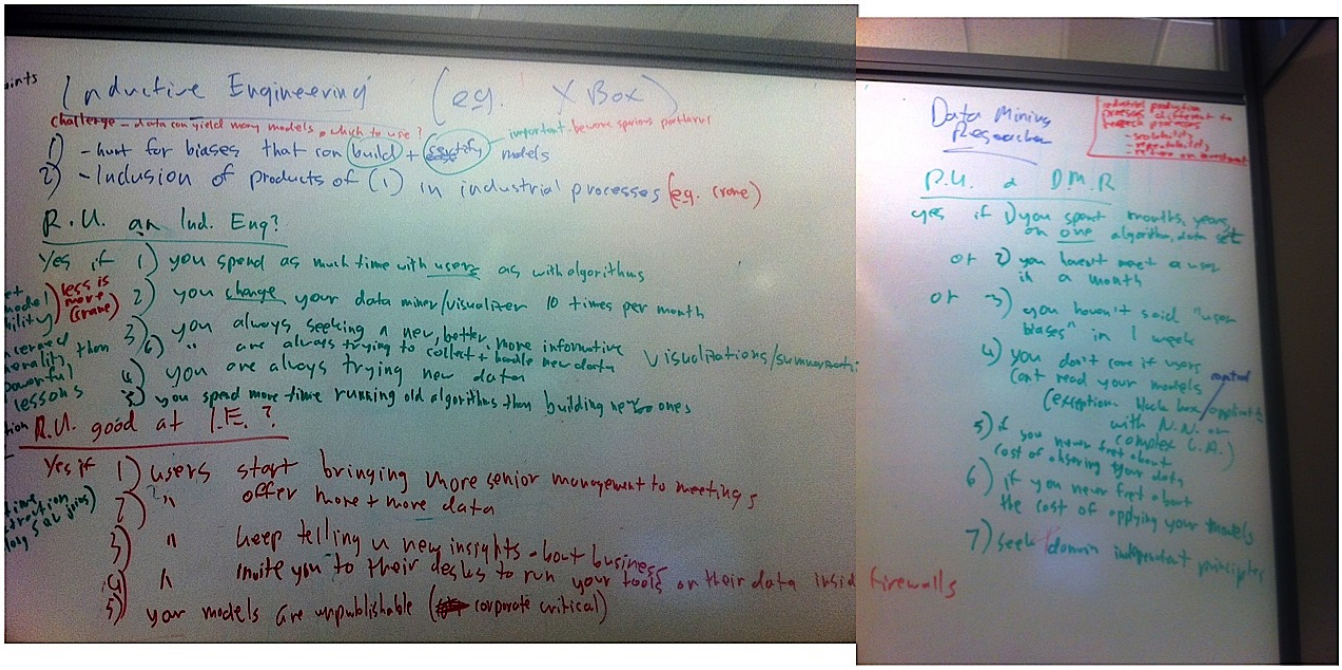

Figure 2. Manifesto, version 0.1.

> *Principle 1:* ***Users before algorithms:*** Mining algorithms are only useful in industry if users fund their use in real-world applications.

The user perspective is vital to inductive engineering. The space of models that can be generated from any data set is very large[1]. If we understand and apply user goals, then we can quickly focus an inductive engineering project on the small set of most crucial issues.

Other researchers have noted the importance of understanding the user perspective. For example, Fenton builds defect models from Bayes models as part of extensive user sessions [19]. In addition, Valerdi & Boehm build effort models via Delphi sessions that combine human judgment with the output of data miners [7], [37], [38]. Based on that work, and our own experience, we assert that:

> *Definition 1:* *Inductive Software Engineering*: Understanding user goals to inductively generate the models that most matter to the user.

We offer the following heuristics to gauge the success of the user interaction meetings. In a good meeting:

- The users keep interrupting to debate the implications of your results. This shows that (a) you are explaining the results in a way they understand; also (b) your results are commenting on issues that concern them.
- The users bring senior management to the meetings.
- The users start listing more and more candidate data sources that you could exploit.
- After the meeting, the users invite you back to their desks inside their firewalls to show them how to perform certain kinds of analysis.

When working with users, it is vital to make the best use of their time. One issue with the user-modeling sessions conducted by Fenton, Valerdi & Boehm is that they require extensive user involvement. For example, Valerdi once recruited 40 experts to three Delphi sessions, each of which ran for three hours. Assuming an 8 hour day, then that study took $3 * 4 * 40/8 = 60\ days$. Therefore, in our work, we try to meet with users weekly for an hour or two. In between meetings, our task is to conduct as much analysis as possible to generate novel insights that interest the user.

III. The Standard KDD Cycle

We are now in a position to comment on the difference of this paper to prior commentaries on how to organize industrial data mining. Fayyad et al. [36] offer the classic definition of data mining, as applied to real-world activities:

> KDD (knowledge discovery in databases) is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data [36].

Figure 3 summarizes their approach. While we take issue with parts of their proposal, many aspects are insightful and worthy of careful study. For example, Figure 3 clearly

[1]And the space of models not supported by the data is much larger. Hence, on some days, we believe that the real role of science in general, or inductive engineering in particular, is to quickly retire unfounded models before they can do any damage.

shows data mining is just a small part of the total process. Even just gaining permission to access data can be a long process requiring extensive interaction with business user groups. Copying large amounts of data from one city to another can also consume a large amount of time. Once data is accessed, then raw data typically requires extensive manipulation before it is suitable for mining. This is:

> *Tip 1: Most of "data mining" is actually "data pre-processing":* Before any learner can execute, much effort must be expended in selecting and accessing the data to process, pre-processing, and transforming it some learnable form.

Figure 3 also clearly illustrates the cyclic nature of inductive engineering:

- Usually, finding one pattern prompts new questions such as "why does that effect hold?" or "are we sure there is no bug in step X of the method?". Each such question refines the goals of the data mining, which leads to another round of the whole process.
- As mentioned above, in the initial stages of a project, engineers try different methods to generate the feedback that let users refine and mature the goals of the project.
- Real world data is highly "quirky" and inductive engineers often try different methods before they discover how to find patterns in the data.

The repetitive nature of inductive engineering implies:

> *Principle 2:* ***Plan for scale:*** In any industrial application, the data mining method is repeated multiples time to either answer an extra user question, make some enhancement and/or bug fix to the method, or to deploy it to a different set of users.

This, in turn, has implications on tool choice:

> *Tip 2: Thou shall not click:* For serious studies, to ensure repeatability, the entire analysis should be automated using some high level scripting language; e.g. R-script, Matlab, or Bash [32].

Figure 3 was highly insightful when it was published in 1996. In 2011, we would we augment it as follows:

A. User involvement;
B. Cycle evolution;
C. Early feedback.

We expand on these points, below.

A. User Involvement

We made the case above that successful inductive software engineering projects require extensive user input. User input is not a primary concern of Figure 3.

B. Cycle Evolution

Our experience is that while inductive software engineering is cyclic, *the cycles evolve as the project matures*. For example, consider the CRANE application developed by inductive engineers at Microsoft [14]. CRANE is a risk assessment and test prioritization tool used at Microsoft that alerts developers if the next software check-in might be problematic. CRANE makes its assessments using metrics collected from static analysis of source code, dynamic analysis of tests running on the system, and field data. CRANE's development process only partially matches Figure 3. Instead, development of CRANE required the following phases:

1) The *scout* phase: an initial rapid prototyping phase where many methods were applied to the available

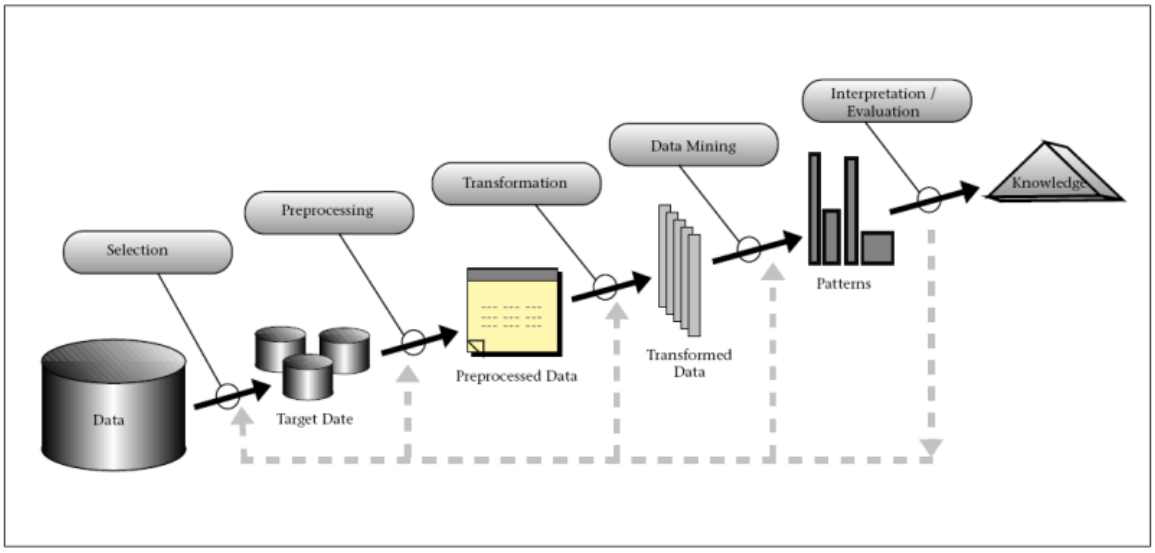

Figure 3. Fayyad et al.'s classic depiction of applying data miners. From [36].

data. In this phase, experimental rigor is less important than exploring the range of user hypotheses. The other goal of this phase is to gain the interest of the users in the induction results.

2) The *survey* phase: After securing some interest and good will amongst the user population, inductive engineers conducted careful experiments focusing on the user goals found during the scouting phase. Of these three phases, the *survey* phase is closest to Figure 3.
3) The *build* phase: After the *survey* has found stable models of interest to the users, a systems engineering phase begins where the learned models are integrated into some turn-key product that is suitable for deployment to a very wide user base.

In terms of development effort, the specific details of CRANE's development schedule are proprietary. We have observed that for *greenfield applications* which have not been developed previously:

- The development effort often takes weeks/ months/ years of work for *scout*/ *survey*/ *build* (respectively).
- The team size doubles and then doubles again after the *scout* and *survey* phases; e.g one scout, two surveyors, and four builders (assisted by the analysis of the surveyors).

For *product line applications* (where the new effort is some extension to existing work), the above numbers can be greatly reduced when the new effort reuses analysis or tools developed in previous applications.

C. Early Feedback

The point of the initial *scout* phase is to get feedback, as early as possible, from the users about appropriate directions for the project. We are continually surprised at how much insight our users gain even from our preliminary pre-processing results. For example, as part of data pre-processing, Dougherty et al. recommend discretization of continuous attributes [16]. Supervised discretizers hunt for divisions to numeric data where the class distributions change the most. Discretization can determine which attributes are *ignorable*. Figure 4 shows one data set where two of the attributes do not split at all; i.e. the discretizer found no value in dividing up certain columns. Such displays can be profoundly insightful to users since it lets them ignore side issues and lets them focus on the most important factors.

Note that discretizers run on log-linear time [18] and, hence, can terminate even when more elaborate data miners fail. Other pre-processors that can offer rapid feedback are linear-time feature selection [23] or instance selection [34] tools. Hence, we recommend:

> *Principle 3:* ***Early feedback:*** Continuous and early feedback from users, allows needed changes to be made as soon as possible (e.g. when they find that assumptions don't match the users' perception) and without wasting heavy up-front investment.

> *Tip 3: Simplicity first:* Prior to conducting very elaborate studies, try applying very simple tools to gain rapid early feedback.

In our work we stress that feedback to the users *can and must appear very early* in a inductive engineering project. Users, we find, find it very hard to express want they want from their data. This is especially true if they have never mined it before. However, once we start showing them results, their requirements rapidly mature as initial results help them sharpen the focus of the inductive study. Hence:

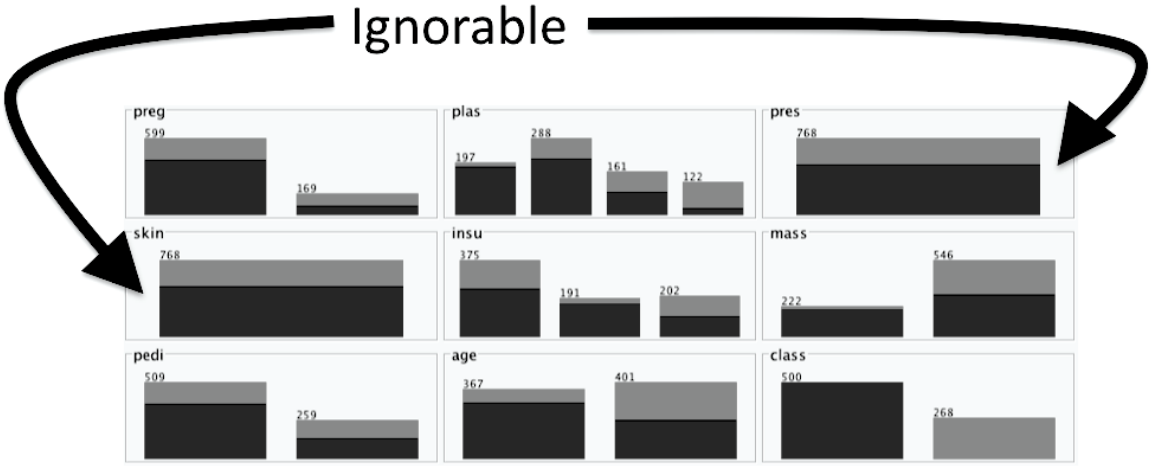

Figure 4. Results of discreting the diabetes data set from the UCI repository; see http://goo.gl/76qYR. Black and gray denote two classes. Vertical bars denote the splits found in eight independent variables. An ignorable column is one that cannot be divided to minimize class diversity. In this data, two columns are ignorable.

Principle 4: ***Be open-minded:*** It is unwise to enter into an inductive study with fixed hypotheses or approaches particularly for data that has not been mined before. Don't resist exploring additional avenues when a particular idea doesn't work out.

> *Tip 4: Data likes to surprise:* Initial results often changes the goals of a study when (e.g.) business plans are based on issues that irrelevant to local data.

IV. Details

Having made our main point about the difference of our proposal to the 1996 version of the KDD cycle, this section discusses some details of inductive engineering.

A. Avoiding Bad Learning

Principle 5: ***Do smart learning:*** Important outcomes are riding on your conclusions. Make sure that you check and validate them.

We stressed above that real-world inductive engineering requires trying a range of methods on each data sets. One danger with trying too many methods is "data dredging"; i.e. the the inappropriate (sometimes deliberately so) use of data mining to uncover misleading relationships in data.

Certainly, if data is tortured enough, it will reveal bogus results [13]. On the other hand, sometimes it is appropriate to conduct a "fishing expedition", just to see what patterns might exist. However, when fishing for patterns, it is wise to maintain a healthy scepticism about the generated conclusions. In this regard, rule-learners like RIPPER [12] and INDUCT [20] are interesting since they check the rules against randomly generated alternatives (and if the probability of a rule is not better than that of random selection, it is deleted). Such tests are not standard in other tools and so we recommend the following:

> *Tip 5: Check the variance:* Before asserting that result A is better than result B, repeat the analysis for multiple subsets of the data and perform some statistical tests to check that any performance differences are not just statistical noise[2]
>
> *Tip 6: Check the stability:* Given any conclusion, see if it holds if the analysis is repeated using (say) 10*90% random samples of data.

The lower the support for the conclusion, the less likely that it will hold across multiple sub-samples. Hence:

> *Tip 7: Check the support:* Try to avoid conclusions based on a very small percent of the data.

[2]Caution: standard tests such ANOVA and t-tests make Gaussian assumptions that do not hold in many domains. Consider the use of non-parametric tests such as Wilcoxon or Mann-Whitney, Friedman/Nemenyi.

One trick for increasing conclusion support is to "chunk up" numeric ranges into a few bins[3], thus preventing the learner from building models using very small numeric ranges.

B. Data and Hypothesis Collection

While the last 3 rules are useful, they do not necessarily prevent data dredging. Making wrong conclusions from observations is a problem for any inductive process. In fact, humans can get induction just as wrong as an automated algorithm. WIKIPEDIA lists 96 known human cognitive biases[4]. As documented by Simons and Chabris [35], the effects of these biases can be quite startling.

Since all inductive agents (be they human or artificial) can make inductive errors, we must employ some methods to minimize the frequency of those errors. The standard solution is use some initial requirements gathering stage where the goals of the learning are defined in a careful and reflective way, as discussed in:

- Basili's Goal-Question-Metric [28] approach;
- Easterbrook et al.'s notes on empirical SE [17].

The intent of this solution is to prevent spurious conclusions by (a) carefully controlling data collection and by (b) focusing the investigation on a very small space of hypotheses. Where possible:

> *Tip 8: If you can, control data collection:* The Goal-Question-Metric approach suggests putting specific data collection measures in place early to address a goal.

The opportunity to use the above tip is rare in practice, and you should take advantage of it when it comes. But one note of caution:

> *Tip 9: Be smart about data collecting and cleaning.:* Keep in mind that collecting data comes at a cost (for example, related to hardware, runtime, and/or operations) and should not negatively affect the users' system (avoid runtime overhead). It is thus infeasible to collect all possible data. Collect data that has high return on investment, i.e., many insights for relatively little cost.

In the usual case, you cannot control data collection. For example, when Menzies worked at NASA, he had to mine information collected from layers of sub-contractors and sub-sub-contractors. Any communication to data owners had to be mediated by up to a dozen account managers, all of whom had much higher priority tasks to perform. Hence, we caution that usually you must:

Principle 6: ***Live with the data you have:*** You go mining with the data you have—not the data you might want or wish to have at a later time.

[3]e.g. divide them at the 33 and 66-th percentile change, or use some supervised discretizer [18].

[4]38 decision making biases; 30 biases in probability; 18 social biases; and 10 memory biases. See http://goo.gl/Z7ij.

That is, the task of the inductive engineering is to make the most of the data at hand, and not wait for some promised future data set that might never arrive. Since we may not have control over how data is collected, it is wise to cleanse the data prior to learning:

> *Tip 10: Rinse before use:* Before learning from a data set, conduct instance or feature selection studies to see what spurious data can be removed. Many tool kits include feature selection tools (e.g. WEKA). As to instance selection, try clustering the data and reasoning from just a small percentage of the data from each cluster. Data cleaning has a cost too, and it is highly unlikely that you can afford perfect data. Most data mining approaches can handle a limited amount of noise.

Note that a standard result is that given a table of data, 80 to 90% of the rows and all but the square root of the number of columns can be deleted before comprising the performance of the learned model [10], [23], [25], [27].

As to controlling the space of hypotheses to be explored, we have often found that

> *Tip 11: Helmuth von Moltke's rule:* Few hypotheses survive first contact with the data.

Even if the users are most concerned about X, Y, the data may be silent on X, only comment on some modified form of Y', but contain significant insights about Z (something the users have never considered before). This is particularly true for data that has not been mined before.

For example, it is clear that product complexity is a critical factor in determining the development cost of software. But this critical factor is an irrelevancy in the NASA93 data set from http://goo.gl/WlzCC where 83% of all the data is labeled "high complexity" [11]. Experiences like the above tell that until we "lift the lid" and look at the actual data, we need to take a respectful, but doubtful, approach to all domain hypothesis offered by the users.

C. Tool Choice

Researchers have the luxury of working on a single algorithm (perhaps for many years). Industrial inductive engineers, on the other hand, may try multiple algorithms each day, to generate some novel and insightful feedback to the users. Hence:

> *Principle 7:* ***Broad skill set, big toolkit:*** Successful inductive engineers routinely try multiple inductive technologies.

To handle the wide range of possible goals of different goals, an inductive engineer should be ready to deploy a wide range of tools. For example, Figure 5 comes from a a survey of the decision making requirements of hundreds of Microsoft developers [8].

	past	present	future
exploration (find)	trends	alerts	forecasting
analysis (explain)	summarization	overlays	goals
experimentation (what-if)	modeling	benchmarking	simulation

Figure 5. Nine kinds of decision making needs found in a sample of industrial practitioners. From [8].

Note that no single data miner supports all parts of Figure 5:

- Regression methods, or data stream miners [21], could detect and track *trends*.
- Anomaly detectors [9] or contrast set learners [30], [33] can *alert* users that old models need to be changed.
- A supervised discretization algorithm can offer an *summarization* of how each variable effects the domain.
- For more detailed *modeling*, learn models showing the interactions between multiple variables in the data.
- Suppose a separate Naive Bayes classifier is built for each cluster in the data [26]. The statistics collected in this way can generate *forecasts* for what to expect if the business moves into a particular cluster.
- Suppose the clusters from the last point are grouped into a dendrogram (a hierarchical cluster tree). Summaries of the data could be generated by examining all the examples that fall into any node of the dendogram:
  - An *overlay* (describing the current state of the data) would just be a display of (e.g.) the mean and variance of the class variable in each node.
  - A comparisons against *benchmark* data would just be a display of the *delta* between *acceptable standard results* and the *mean performance score*.
  - To understand how close the data comes to desired *goals*, display the *delta* between *stated goals* and the *performance score mean*.
- Finally, in this framework, *simulation* is just pushing what-if queries into the models learned via data mining, and seeing what effects they generate.

Note that the set of useful inductive technologies is large and constantly changing. To have access to the cutting edge of data mining tools:

> *Tip 12: Big ecology:* Use tools supported by a large ecosystem of developers who are constantly building new learners and fixing old ones; e.g. R, WEKA, MATLAB.

V. Implications

A. Implications for Project Management

We listed above our preferred approach to data mining applications: *scout* (initial tentative conclusions); *survey* (more careful explorations); then *build* (standard scale-up).

Our experience tells us that *scouting, surveying*, and *building* takes weeks, months, and years (respectively).

B. Implications for Training

The current set of data mining/ machine learning/ pattern recognition classes are excellent at training academic data miners. That said, it would be useful to augment those classes with inductive engineering project work (either as part of those classes or as a capstone project or as a separate inductive inductive engineering class). Such projects could structure their training around the *scout* and *survey* stage. For example, at Menzies' WVU graduate data mining classes, students spend 7 weeks learning basic tools. This is followed by a three week *scout* project where each week, they are required to report back to the class something profound in their data. To meet such a tight schedule, they will by necessity have to cut corners as they race to generate preliminary results. One deliverable of that *scout* project is a list of all the short cuts they made in their analysis. This becomes a specification of a final five week *survey* project where they repeat their entire analysis, much slower, much more thoroughly.

We stressed above the need to present results to users, to gain their feedback. Since communication is such an essential skill for an inductive engineer, we recommend that student training also includes written report generation as well as presenting that material in a briefing.

Another requirement we would add to training programs is the need to teach the scripting skills needed to automate some data mining analysis.

C. Implications for Academic Research

From the above, we can isolate research themes that might most benefit industrial data mining:

- *Analysis patterns of inductive engineers:* Skilled engineers make better use of the data miners than novices. It would be useful to document the patterns of the expert users and the anti-patterns of the novices.
- *Design patterns for data miners:* The tool kits we use constantly change. It would be useful to be able to easily and quickly maintain and extend them.
- *Optimizations of learning algorithms:* To better support the "scout" phase, it would be preferable to have faster, more scalable learners.
- *Anomaly detectors:* It would be useful to have automatic detectors that alert us when the models learned from scouting or surveying need revisions and/or updating.
- *Business-aware learners:* It would be useful to be able to quickly adjust the biases of our learners towards the user biases. For preliminary notes on that kind of work, see the WHICH learner described in [31].
- Instead of viewing data mining as a "one-shot" process, we characterize it as incremental exploration, with the assistance of the user. Research on the following areas would assist such an exploration:
  - *Visualization:* To support *scout*ing, it would be useful to have better visualizations of data and the learned models. Such learned models are very useful when offering insights to users.
  - *Anytime learning:* An anytime algorithm can offer, at any time, a working result. Also, at any future time, it can offer a better result. Anytime learners would be useful for incremental exploration.
  - *Active learning:* Active learning assumes that some examples are more informative than the others when building a learner [15], [24]. It can be defined as a learning process, where a learner is given a data set without labels and some oracle can label the instances upon request. Ideally, the learner discovers as few labels as possible by choosing the most informative instances that are most beneficial for the learning process. Active learning would reduce the time required by users involved in the incremental exploration.

VI. Conclusion

This article has listed our conclusions from decades of combined work into data mining. We hope it has demonstrated that, in the field of inductive software engineering, there are important generalities which we can, and should, share. Such a pooling of knowledge is essential to maturing an engineering profession since that knowledge:

- Defines certification criteria for new inductive engineers;
- Improves our ability to recruit new hires to a site performing inductive engineering (since we will have a clearer understanding of the skill set we need to hire);
- Allow training organizations (universities and private consultancy companies) to create better, more industry relevant, training programs.

We look forward to a rapid evolution of this manifesto as more inductive engineers (a) discuss their methods and (b) find common themes in their work.

References

[1] How do practitioners perceive software engineering research?

[2] James H. Andrews, Tim Menzies, and Felix C.H. Li. Genetic algorithms for randomized unit testing. *IEEE Transactions on Software Engineering*, March 2010. Available from http://menzies.us/pdf/10nighthawk.pdf.

[3] E. Barr, C. Bird, E. Hyatt, Tim Menzies, and G. Robles. On the shoulders of giants. In *FoSER 2010*, November 2010. Available from http://menzies.us/pdf/10giants.pdf.

[4] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*, MSR '06, pages 137–143, 2006.

[5] Christian Bird, Brendan Murphy, Nachiappan Nagappan, and Thomas Zimmermann. Empirical software engineering at microsoft research. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, CSCW '11, pages 143–150, 2011.

[6] Christian Bird, Nachiappan Nagappan, Premkumar Devanbu, Harald Gall, and Brendan Murphy. Does distributed development affect software quality?: an empirical case study of windows vista. *Commun. ACM*, 52:85–93, August 2009.

[7] Barry Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[8] Raymond P L Buse and Thomas Zimmermann. Information needs for software development analytics. *MSR-TR-2011-8*, 2011.

[9] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:15:1–15:58, July 2009.

[10] C.L. Chang. Finding prototypes for nearest neighbor classifiers. *IEEE Trans. on Computers*, pages 1179–1185, 1974.

[11] Zhihoa Chen, Tim Menzies, and Dan Port. Feature subset selection can improve software cost estimation. In *PROMISE'05*, 2005. Available from http://menzies.us/pdf/05/fsscocomo.pdf.

[12] W.W. Cohen. Fast effective rule induction. In *ICML'95*, pages 115–123, 1995. Available on-line from http://www.cs.cmu.edu/~wcohen/postscript/ml-95-ripper.ps.

[13] R.E. Courtney and D.A. Gustafson. Shotgun correlations in software measures. *Software Engineering Journal*, 8(1):5 –13, jan 1993.

[14] J. Czerwonka, R. Das, N. Nagappan, A. Tarvo, and A. Teterev. Crane: Failure prediction, change analysis and test prioritization in practice – experiences from windows. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pages 357 –366, march 2011.

[15] Sanjoy Dasgupta and Daniel Hsu. Hierarchical sampling for active learning. *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 208–215, 2008.

[16] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *International Conference on Machine Learning*, pages 194–202, 1995. Available from http://www.cs.pdx.edu/~timm/dm/dougherty95supervised.pdf.

[17] S.M. Easterbrook, J. Singer, M. Storey, and D. Damian. Selecting empirical methods for software engineering research. In F. Shull and J. Singer, editors, *Guide to Advanced Empirical Software Engineering*. Springer, 2007.

[18] U M Fayyad and I H Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022–1027, 1993.

[19] Norman Fenton, Martin Neil, William Marsh, Peter Hearty, Lukasz Radlinski, and Paul Krause. Project data incorporating qualitative factors for improved software defect prediction. In *PROMISE'09*, 2007. Available from http://promisedata.org/pdf/mpls2007FentonNeilMarshHeartyRadlinskiKrause.pdf.

[20] B.R. Gaines and P. Compton. Induction of ripple down rules. In *Proceedings, Australian AI '92*, pages 349–354. World Scientific, 1992.

[21] Joao Gama and Carlos Pinto. Discretization from data streams: applications to histograms and data mining. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 662–667, New York, NY, USA, 2006. ACM Press. Available from http://www.liacc.up.pt/~jgama/IWKDDS/Papers/p6.pdf.

[22] Gregory Gay, Tim Menzies, Misty Davies, and Karen Gundy-Burlet. Automatically finding the control variables for complex system behavior. *Automated Software Engineering*, (4), December 2010. Available from http://menzies.us/pdf/10tar34.pdf.

[23] M.A. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions On Knowledge And Data Engineering*, 15(6):1437– 1447, 2003. Available from http://www.cs.waikato.ac.nz/~mhall/HallHolmesTKDE.pdf.

[24] M Kääriäinen. Active learning in the non-realizable case. *Algorithmic Learning Theory*, 2006.

[25] E. Kocaguneli, T. Menzies, A. Bener, and J. Keung. Exploiting the essential assumptions of analogy-based effort estimation. *IEEE Transactions on Software Engineering*, 2011 (preprint). Available from http://menzies.us/pdf/11teak.pdf.

[26] R. Kohavi, D. Sommerfield, and J. Dougherty. Data minining using mlc++: A machine learning library in c++. In *Tools with AI 1996*, 1996.

[27] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[28] Yasuhiro Mashiko and Victor R. Basili. Using the gqm paradigm to investigate influential factors for software process improvement. *Journal of Systems and Software*, 36:17–32, 1997.

[29] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, January 2007. Available from http://menzies.us/pdf/06learnPredict.pdf.

[30] Tim Menzies and Y. Hu. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from http://menzies.us/pdf/03tar2.pdf.

[31] Tim Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener. Defect prediction from static code features: Current results, limitations, new approaches. *Automated Software Engineering*, (4), December 2010. Available from http://menzies.us/pdf/10which.pdf.

[32] Adam Nelson, Tim Menzies, and Gregory Gay. Sharing experiments using open-source software. *Softw. Pract. Exper.*, 41:283–305, March 2011.

[33] Petra Kralj Novak, Nada Lavrač, and Geoffrey I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *J. Mach. Learn. Res.*, 10:377–403, June 2009.

[34] J. Arturo Olvera-López, J. Ariel Carrasco-Ochoa, J. Francisco Martínez-Trinidad, and Josef Kittler. A review of instance selection methods. *Artif. Intell. Rev.*, 34:133–143, August 2010.

[35] D.J. Simons and C.F. Chabris. Gorillas in our midst: Sustained inattentional blindless for dynamic events perception. *Perception*, 28:1059–1074, 1999.

[36] Gregory Piatetsky-Shapiro Usama Fayyad and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, pages 37–54, Fall 1996.

[37] Miller C.-Thomas G. Valerdi, R. Systems engineering cost estimation by consensus. In *17th International Conference on Systems Engineering*, September 2004.

[38] R. Valerdi. Convergence of expert opinion via the wideband delphi method: An application in cost estimation models. In *Incose International Symposium, Denver, USA*, 2011. Available from http://goo.gl/Zo9HT.

[39] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy. Cross-project defect prediction. In *ESEC/FSE'09*, August 2009.

[40] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schrter, and Cathrin Weiss. What makes a good bug report? *IEEE Transactions on Software Engineering*, 36(5):618–643, September 2010.