Seven principles of inductive software engineering: What we do is different

T. Menzies
North Carolina State University, Raleigh, NC, United States

CHAPTER OUTLINE

Different and Important 13
Principle #1: Humans Before Algorithms 13
Principle #2: Plan for Scale 14
Principle #3: Get Early Feedback 15
Principle #4: Be Open Minded 15
Principle #5: Be Smart with Your Learning 15
Principle #6: Live with the Data You Have 16
Principle #7: Develop a Broad Skill Set That Uses a Big Toolkit 17
References 17

DIFFERENT AND IMPORTANT

Inductive software engineering is the branch of software engineering focusing on the delivery of data-mining based software applications. Within those data mines, the core problem is *induction*, which is the extraction of small patterns from larger data sets. Inductive engineers spend much effort trying to understand business goals in order to inductively generate the models that matter the most.

Previously, with Christian Bird, Thomas Zimmermann, Wolfram Schulte, and Ekrem Kocaganeli, we wrote an *Inductive Engineering Manifesto* [1] that offered some details on this new kind of engineering. The whole manifesto is a little long, so here I offer a quick summary. Following are seven key principles which, if ignored, can make it harder to deploy analytics in the real world. For more details (and more principles), refer to the original document [1].

PRINCIPLE #1: HUMANS BEFORE ALGORITHMS

Mining algorithms are only good if humans find their use in real-world applications. This means that humans need to:

- understand the results
- understand that those results add value to their work.

Accordingly, it is strongly recommend that once the algorithms generate some model, then the inductive engineer *talks to humans* about those results. In the case of software analytics, these humans are the subject matter experts or business problem owners that are asking you to improve the ways they are generating software.

In our experience, such discussions lead to a second, third, fourth, etc., round of learning. To assess if you are talking in "the right way" to your humans, check the following:

- Do they bring their senior management to the meetings? If yes, great!
- Do they keep interrupting (you or each other) and debating your results? If yes, then stay quiet (and take lots of notes!)
- Do they indicate they understand your explanation of the results? For example, can they correctly extend your results to list desirable and undesirable implications of your results?
- Do your results touch on issues that concern them? This is *easy* to check… just count how many times they glance up from their notes, looking startled or alarmed.
- Do they offer more data sources for analysis? If yes, they like what you are doing and want you to do it more.
- Do they invite you to their workspace and ask you to teach them how to do XYZ? If yes, this is a real win.

PRINCIPLE #2: PLAN FOR SCALE

Data mining methods are usually repeated multiple times in order to:

- answer new questions, inspired by the current results;
- enhance data mining method or fix some bugs; and
- deploy the results, or the analysis methods, to different user groups.

So that means that, if it works, you will be asked to do it again (and again and again). To put that another way, *thou shalt not click*. That is, if all your analysis requires lots of pointing-and-clicking in a pretty GUI environment, then you are definitely *not* planning for scale.

Another issues is that as you scale up, your methods will need to scale up as well. For example, in our *Manifesto* document, we discussed the CRANE project at Microsoft that deployed data mining methods to the code base of Microsoft Windows. This was a *very large* project, so the way it started was *not* the way it ended:

- Initially, a single inductive engineer did some rapid prototyping for a few weeks, to explore a range of hypotheses and gain business interest (and get human feedback on the early results).

- Next, the inductive engineering team spent a few months conducting many experiments to find stable models (and to narrow in on the most important business goals).
- In the final stage, which took a year, the inductive engineers integrated the models into a deployment framework that was suitable for a target user base.

Note that the team size doubled at each stage—so anyone funding this work needs to know that increasingly useful conclusions can be increasingly expensive.

PRINCIPLE #3: GET EARLY FEEDBACK

This is mentioned previously, but it is worth repeating. Before conducting very elaborate studies (that take a long time to reach a conclusion), try applying very simple tools to gain rapid early feedback.

So, simplicity first! Get feedback early and often! For example, there are many linear time discretizers for learning what are good divisions of continuous attributes (eg, the Fayyad-Irani discretizer [2]). These methods can also report when breaking up an attribute is *not* useful since that attribute is not very informative. Using tools like these, it is possible to discover what attributes can be safely ignored (hint: usually, it's more than half).

PRINCIPLE #4: BE OPEN MINDED

The goal of inductive engineering for SE is to find better ideas than what was available when you started. So if you leave a data mining project with the same beliefs as when you started, you really wasted a lot of time and effort. Hence, some mantras to chant while data mining are:

- Avoid a fixed hypothesis. Be respectful but doubtful of all human-suggested domain hypotheses. Certainly, explore the issues that they raise, but also take the time to look further afield.
- Avoid a fixed approach for data mining (eg, just using decision trees all the time), particularly for data that has not been mined before.
- The most important initial results are the ones that radically and dramatically improve the goals of the project. So seek important results.

PRINCIPLE #5: BE SMART WITH YOUR LEARNING

Let's face it, any inductive agents (human or otherwise) have biases that can confuse the learning process. So don't torture the data to meet preconceptions (that is, it is ok to go "fishing" to look for new insights).

It also true that any inductive agent (and this includes you) can make mistakes. If organizations are going to use your results to change policy, the important outcomes are riding on your conclusions. This means you need to check and validate your results:

- Ask other people to do code reviews of your scripts.
- Check the conclusion stability against different sample policies. For example:
  - Policy1: divide data into (say) ten 90% samples (built at random). How much do your conclusions change across those samples?
  - Policy2: sort data by collection date (if available). Learn from the far past, the nearer past, then the most recent data. Does time change your results? Is time *still* changing your results? It is important to check.
  - Policy3,4,5, etc.: Are there any other natural divisions of your data (eg, east coast versus west coast, men versus women, etc.)? Do they affect your conclusions?
- When reporting multiple runs of a learner, don't just report mean results—also report the wriggle around the mean.
- In fact, do not report mean results at all since outliers can distort those mean values. Instead, try to report median and IQR results (the inter-quartile range is the difference between the 75th and 25th percentile).

PRINCIPLE #6: LIVE WITH THE DATA YOU HAVE

In practice, it is a rare analytics project that can dictate how data is collected in industrial contexts. Usually, inductive engineers have to cope with whatever data is available (rather than demand more data collected under more ideal conditions). This means that often you have to go mining with the data you have (and not the data you hope to have at some later date). So it's important to spend some time on data quality operators. For example:

- Use *feature selection* to remove spurious attributes. There are many ways to perform such feature selection, including the Fayyad-Irani method discussed herein. For a discussion of other feature selection methods, see [3].
- Use *row selection* to remove outliers and group-related rows into sets of clusters. For a discussion on row selection methods, see [4].

One benefit of replacing rows with clusters is that any signal that is spread out amongst the rows can be "amplified" in the clusters. If we cluster, then learn one model per cluster, then the resulting predictions have better median values and smaller variances [5].

In any case, we've often found row and feature selection discards to be up to 80% to 90% of the data, without damaging our ability to learn from the data. This means that ensuring quality of *all* the data can sometimes be less important that being able to extract quality data from large examples.

PRINCIPLE #7: DEVELOP A BROAD SKILL SET THAT USES A BIG TOOLKIT

The reason organizations need to hire inductive engineers is that they come equipped with a very broad range of tools. This is important since many problems need specialized methods to find good solutions.

So, to become an inductive engineer, look for the “big ecology” toolkits where lots of developers are constantly trying out new ideas. Languages like Python, Scala (and lately, Julia) have extensive online forums where developers share their data mining tips. Toolkits like R, MATLAB, and WEKA are continually being updated with new tools.

What a great time to be an inductive engineer! So much to learn, so much to try. Would you want to have it any other way?

REFERENCES

[1] Menzies T, Bird C, Zimmermann T, Schulte W, Kocaganeli E. The inductive software engineering manifesto: principles for industrial data mining. In: Proceedings of the international workshop on machine learning technologies in software engineering (MALETS ’11). New York. NY, USA: ACM; 2011. p. 19–26. http://dx.doi.org/10.1145/2070821.2070824.

[2] Fayyad UM, Irani KB. On the handling of continuous-valued attributes in decision tree generation. Mach Learn 1992;8(1):87–102. http://dx.doi.org/10.1023/A:1022638503176.

[3] Hall MA, Holmes G. Benchmarking attribute selection techniques for discrete class data mining. IEEE Trans Knowl Data Eng 2003;15(6):1437–47. http://dx.doi.org/10.1109/TKDE.2003.1245283.

[4] Olvera-López JA, Carrasco-Ochoa JA, Martínez-Trinidad JF, Kittler J. A review of instance selection methods. Artif Intell Rev 2010;34(2):133–43. http://dx.doi.org/10.1007/s10462-010-9165-y.

[5] Menzies T, Butcher A, Cok D, Marcus A, Layman L, Shull F, Turhan B, Zimmermann T. Local versus global lessons for defect prediction and effort estimation. IEEE Trans Softw Eng 2013;39(6):822–34. http://dx.doi.org/10.1109/TSE.2012.83.