Empiricism Is Not a Matter of Faith

Ted Pedersen[*]
University of Minnesota, Duluth

1. The Sad Tale of the Zigglebottom Tagger

"Hurrah, this is it!" you exclaim as you set down the most recent issue of *Computational Linguistics*. "This Zigglebottom Tagger is exactly what I need!" A gleeful smile crosses your face as you imagine how your system will improve once you replace your tagger from graduate school with the clearly superior Zigglebottom method. You rub your hands together and page through the article looking for a way to obtain the tagger, but nothing is mentioned. That doesn't dampen your enthusiasm, so you search the Web, but still nothing turns up. You persist though; those 17 pages of statistically significant results really are impressive. So you e-mail Zigglebottom asking for the tagger.

Some days, or perhaps weeks, later, you get a hesitant reply saying: "We're planning to release a demo version soon, stay tuned . . . " Or perhaps: "We don't normally do this, but we can send you a copy (informally) once we clean it up a bit . . . " Or maybe: "We can't actually give you the tagger, but you should be able to re-implement it from the article. Just let us know if you have any questions . . . "

Still having faith, and lacking any better alternative, you decide to re-implement the Zigglebottom Tagger. Despite three months of on-and-off effort, the end result provides just the same accuracy as your old tagger, which is nowhere near that reported in the article. Feeling sheepish, you conclude you must have misunderstood something, or maybe there's a small detail missing from the article. So you contact Zigglebottom again and explain your predicament. He eventually responds: "We'll look into this right away and get back to you . . . "

A year passes. You have the good fortune to bump into Zigglebottom at the Annual Meeting of the Association for Computational Linguistics (ACL). You angle for a seat next to him during a night out, and you buy him a few beers before you politely resume your quest for the tagger. Finally, he confesses rather glumly: "My student Pifflewhap was the one who did the implementation and ran the experiments, and if he'd only respond to my e-mail I could ask him to tell you how to get it working, but he's graduated now and is apparently too busy to reply."

After a few more beers, Zigglebottom finally agrees to give you the tagger: "I'll send you the version of the code I have, no promises though!" And true to his word, what he sends is incomplete and undocumented. It doesn't compile easily, and it's engineered so that a jumble of programs must be run in an undisclosed kabalistic sequence known only to (perhaps) the elusive Pifflewhap. You try your best to make it work every now

[*] Department of Computer Science, 1114 Kirby Drive, University of Minnesota, Duluth, MN 55812, USA. E-mail: tpederse@d.umn.edu.

and then for a few months, but eventually you give up, and go back to using the same old tagger you used before.

2. The Paradox of Faith-Based Empiricism

The tale of the Zigglebottom Tagger is one of disappointment, not just for you but also for Zigglebottom himself. While his work achieved publication, it must gnaw at his scientific conscience that he can't reproduce his own results. The fact that you can't reproduce those results either raises questions, but those are resolved with a shrug of your shoulders and by giving the benefit of the doubt to Zigglebottom. He's not a fraud; there's just some crucial detail that is neither recorded in the article nor in the software, which can't be installed and run in any case.

The problem here is not the Zigglebottom article; as a community we accept that our publications don't provide enough space to describe our elaborate 21st century empirical methods in sufficient detail to allow for re-implementation and reproduction of results. This is true despite the generous page allowances in *Computational Linguistics* and even more so in our much more constrained conference proceedings.

What's really missing is the software that produced the results that convinced the reviewers the article should be published. This is particularly troubling given the highly empirical nature of the work reported in so many of our publications. We publish page after page of experimental results where apparently small differences determine the perceived value of the work. In this climate, convenient reproduction of results establishes a vital connection between authors and readers.

Our community expects published papers to be rigorously reviewed and made available via open access as soon as possible (e.g., via the ACL Anthology[1]). We expect the supporting corpora and lexical resources will be made available even if at some cost (e.g., via the Linguistic Data Consortium[2]). Yet, we do not have the same expectations regarding our software. While we have table after table of results to pore over, we usually don't have access to the software that would allow us to reproduce those results. This cuts to the core of whether we are engaged in science, engineering, or theology: Scientists reproduce results; engineers build impressive and enduring artifacts; and theologians muse about what they believe but can't see or prove.

Before you judge the analogy with theology as being too harsh, conduct the following experiment. Randomly select one of your own publications from a year or two ago and think about what would be involved in reproducing the results. How long would it take, assuming you would be able to do it? If you can't reproduce those results, why do you believe them? Why should your readers?

Our inability to reproduce results leads to a debilitating paradox, where we as reviewers and readers accept highly empirical results on faith. We do this routinely, to the point where we seem to have given up on the idea of being able to reproduce results. This is the natural consequence of faith-based empiricism, and the only way to fight that movement is with a little bit of heresy. Let's not accept large tables of empirical results on faith, let's insist that we be able to reproduce them exactly and conveniently. Let's insist that we are scientists first and foremost, and agree that this means that we must be able to reproduce each other's results.

1 www.aclweb.org/anthology/.
2 www.ldc.upenn.edu/.

3. A Heretic's Guide to Reproducibility

In many cases the failure to release software that allows results to be reproduced is not a conscious decision, but rather unintentional fallout from how we manage projects and set priorities in guiding our careers. What follows are a few simple ideas that any researcher can adopt to make it much easier (and more likely) to produce software that can not only be released but that will allow users to reproduce results with minimal effort. As more of us use and release such software, our expectations as a community will rise, and we'll eventually see software releases as a natural part of the publication process, much as we now view data sharing.

3.1 Release Early, Release Often

The single greatest barrier to releasing software is that we don't think about doing it early enough. It's only when we get that first e-mail asking for the implementation of a method discussed in *Computational Linguistics* that the issue arises, and by then it's too late. At that point the task of converting our code into a well-documented and easy to use package is often nearly impossible.

Beyond difficulties caused by poor documentation, the passage of time, and turnover in project members, there can even be legal concerns. When projects do not plan to release software, it's often the case that system development will include stages based on helter-skelter cutting and pasting of code from other sources. The effect of this is to erase all traces of the origin of that code and the terms under which it was made available. Once you have gone down this route, it's very hard to consider releasing the resulting software.

However, if you plan from the start to distribute your software, you will inevitably be guided by considerations that are important to your potential audience. You will choose licenses, hardware platforms, and programming languages that avoid any obvious barriers to distribution and use. You will develop an infrastructure of Web services, software repositories, and mailing lists that will evolve with your project. You will avoid haphazard development methodologies that lead to disorganized and impossible-to-maintain code. The prospect of having actual external users of your software will inspire a discipline and orderliness on your development and deployment processes that will likely result in much better software than if you developed it for internal use only.

It is true that releasing software that is both usable and reliable requires a strong hand to guide system development, and that's a skill that many researchers don't *think* they have. However, it's really quite simple to develop. All you must do is play the part of a demanding yet naive client from time to time from the very start of the project. Insist that the code be easy to install and use and that the results that come from it be easy to understand and absolutely reproducible. If the project is too large for you to play this role yourself, assign it to one or more members of your team, and make sure they play the part as if they are a new user encountering the system for the first time.

If you do this from the beginning of a project it takes surprisingly little time, and you end up with much better documentation and software, and a system that can be easily and conveniently used to reproduce results both by outside users and by yourself after the passage of some time.

3.2 Measure Your Career in Downloads and Users

Researchers sometimes fall into the trap of seeing software and reproduction of results as frills, and not essential components in their career development: "As much as I would like to, I don't have the time to produce distributable code. Besides, my promotion will be based on publications and grants, not software releases . . . "

This suggests that you can either spend your time creating and releasing software, or you can spend it writing grant proposals and papers, but not both. This overlooks a very happy side-effect that comes from creating releasable code—you will be more efficient in producing new work of your own since you can easily reproduce and extend your own results.

There is also a danger that this attitude will evolve over time into a self-perpetuating cycle: "I've worked on this for X years, why would I just give it away?" This ignores the fact that "giving it away" will make it easier for others to use your work, because if you don't make your code available, who is really going to spend years re-implementing what you did?

Webber (2007) draws attention to the amount of time our community wastes in writing and reviewing papers that are rejected and eventually abandoned. In a similar vein, we should all think about the time we cost our community when we don't release software and make anyone who is interested in using or validating our work do their own implementation.

If software is released publicly under one of the standard licenses that protects your copyright (e.g., the GNU General Public License[3] or the Mozilla Public License[4]) then there is little danger of your work being misappropriated, and you will build a reservoir of good will within our community. Most users don't want to steal from you; they simply want to use your code to build their own system while giving you all the credit that is your due. As your software acquires a following, you can use that as a foundation for offering tutorials and workshops and other means of dissemination that will increase your visibility in the research community, thereby enhancing the credibility and impact of the work you have done.

3.3 Ensure Project Survivability By Releasing Software

Released software can allow your project to sustain itself despite turnover in personnel and the passage of time. There is no greater satisfaction than opening up a software release that has not been used for a few years and immediately being able to start producing meaningful results, without having to reverse engineer it or trace through code line by line. The more time passes, the more you become just like every other potential user of your software; so, as you are creating it, remember that in a few years your memory of all the details that now seem so obvious will have faded, and you will be grateful for a job well done, and that will translate into time saved as you begin to use that software again.

Imagine meeting with a new project member and being able to say: "Go download this software, read the documentation, install it, run the script that reproduces our ACL experiments, and then we can start talking tomorrow about how you are going to extend that work . . . " This lowers the bar for entry to your project for new colleagues, and saves

3 www.gnu.org/copyleft/gpl.html.
4 www.mozilla.org/MPL/.

your existing team considerable time when introducing a new member to the work of your group.

Although you won't spend much time thinking about it at the start of a project, your students will graduate, post-docs will move on, employees will resign, and you might even find a better job somewhere. Having publicly released software helps clarify what rights former project members have once they have left a project. This is a painfully murky area, and it can lead to many misunderstandings and bad feelings that take time and energy to deal with as they arise.

That confusion can also cause former colleagues to distance themselves from a project simply because they feel they don't have the right to participate, and in fact in some cases they may not even have access to or copies of the very system they spent all those months or years working on. This difficult situation is absolutely avoided if you release the software: Your former colleagues will have exactly the same rights as anyone else. They can remain a part of the community of users, testers, and developers, and can often provide valuable continuity in a project even if they have moved to a new project or organization. The same is true for you. Suppose you move from the academic world to a position in industry: If your project code has already been released prior to this move, then you can safely continue to use it without fear of losing control of it to your new employer.

3.4 Make The World A Better Place

Finally, although this viewpoint may seem quaint or naive, a great deal of our research is funded by public tax dollars, by people who make ten dollars an hour waiting tables or standing behind a counter in a convenience store for 12 hours at a time. We are fortunate to do what we do: even if it takes many hours and causes great personal stress, in the end the work is challenging and satisfying, and compared to how most people in the world live and work, we are leading charmed and privileged lives.

Although most taxpayers won't have much interest in reading our papers and running our code, they ought to have that opportunity. And who knows, maybe when their children take a Computational Linguistics or Artificial Intelligence class they will run across a piece of our publicly available code that will cause them to pause and think, and maybe inspire them to try something new or different, maybe even make them think about becoming one of our community. It's not the most likely scenario, but it seems like we really ought to try to give back as much as we can to the greater public good.

4. What should *Computational Linguistics* Do?

We seem as a community to have accepted a very curious state of affairs. As reviewers and readers of *Computational Linguistics* and the proceedings of ACL conferences, we insist upon extensive, rigorous, and fine-grained evaluations, where the difference in performance between competing methods is sometimes rather small. However, we don't expect to be able to reproduce these results or modify these experiments in any way.

With the rise of search engines as a source of linguistic data, we may have even reached a point where we don't expect our data to be reproducible due to the arbitrary results they provide. Kilgarriff (2007) argues, "Googleology is bad science," to which we would simply add "because it is not reproducible."

But instead of insisting upon reproducibility, we tell ourselves to think about the bigger picture, to focus on the ideas and not the software, as those are just "implementation issues." This is a debilitating paradox, because results must be supported experimentally with great precision and detail and are judged according to harsh empirical standards, but we as readers and reviewers are asked to accept that these results are accurate and reproducible on faith.

If we believe in empirical methods and the value of comparisons and experimental studies, then we must also believe in having access to the software that produced those results as a necessary and essential part of the evidentiary process. Without that we are asked to re-implement methods that are often too complicated and underspecified for this to be possible, or to accept the reported results as a matter of faith.

There are two courses of action open to us. One is to back away from the very stringent standards that focus on evaluation and comparisons of empirical results; to approach things more with a focus on bigger ideas, and less on statistically significant empirical results. This is not necessarily a bad thing, and might address concerns such as those raised by Chuch (2005) about very conservative reviewing in our field and the resulting tendency to prefer incremental improvements.

However, the other path is to accept (and in fact insist) that highly detailed empirical studies must be reproducible to be credible, and that it is unreasonable to expect that reproducibility be possible based on the description provided in a publication. Thus, releasing software that makes it easy to reproduce and modify experiments should be an essential part of the publication process, to the point where we might one day only accept for publication articles that are accompanied by working software that allows for immediate and reliable reproduction of results.

Acknowledgments
I would like to thank Robert Dale for suggesting this topic, and for his many helpful comments and suggestions.

References

Chuch, Kenneth. 2005. Reviewing the reviewers. *Computational Linguistics*, 31(4):575–578.

Kilgarriff, Adam. 2007. Googleology is bad science. *Computational Linguistics*, 33(1):147–151.

Webber, Bonnie. 2007. Breaking news: Changing attitudes and practices. *Computational Linguistics*, 33(4):607–611.