Journal of Machine Learning Research 8 (2007) 2443-2466 Submitted 7/07; Published 10/07

The Need for Open Source Software in Machine Learning

Sören Sonnenburg* SOEREN.SONNENBURG@FIRST.FRAUNHOFER.DE
Fraunhofer Institute FIRST
Kekulestr. 7
12489 Berlin, Germany

Mikio L. Braun* MIKIO@CS.TU-BERLIN.DE
Technical University Berlin
Franklinstr. 28/29
10587 Berlin, Germany

Cheng Soon Ong* CHENGSOON.ONG@TUEBINGEN.MPG.DE
Friedrich Miescher Laboratory
Max Planck Society
Spemannstr. 39
72076 Tübingen, Germany

Samy Bengio BENGIO@GOOGLE.COM
Google
1600 Amphitheatre Pkwy, Building 47-171D
Mountain View, CA 94043, USA

Leon Bottou LEON@BOTTOU.ORG
NEC Laboratories America, Inc.
4 Independence Way Suite 200
Princeton NJ 08540 , USA

Geoffrey Holmes GEOFF@CS.WAIKATO.AC.NZ
Department of Computer Science
University of Waikato
Hamilton, New Zealand

Yann LeCun YANN@CS.NYU.EDU
New York University
715 Broadway
New York, NY 10003, USA

Klaus-Robert Müller KRM@CS.TU-BERLIN.DE
Technical University Berlin
Franklinstr. 28/29
10587 Berlin, Germany

Fernando Pereira PEREIRA@CIS.UPENN.EDU
University of Pennsylvania
3330 Walnut Street
Philadelphia, PA 19104, USA

Carl Edward Rasmussen CER54@CAM.AC.UK
Department of Engineering
Trumpington Street
Cambridge, CB2 1PZ, United Kingdom

*. The first three authors contributed equally.

Gunnar Rätsch GUNNAR.RAETSCH@TUEBINGEN.MPG.DE
Friedrich Miescher Laboratory
Max Planck Society
Spemannstr. 39
72076 Tübingen, Germany

Bernhard Schölkopf BS@TUEBINGEN.MPG.DE
Max Planck Institute for Biological Cybernetics
Spemannstr. 38
72076 Tübingen, Germany

Alexander Smola ALEX.SMOLA@GMAIL.COM
Australian National University and NICTA
Canberra, ACT 0200, Australia

Pascal Vincent VINCENTP@IRO.UMONTREAL.CA
Université de Montréal
Dept. IRO, CP 6128, Succ. Centre-Ville
Montréal, Québec, Canada

Jason Weston JASONW@NEC-LABS.COM
NEC Laboratories America, Inc.
4 Independence Way Suite 200
Princeton NJ 08540 , USA

Robert C. Williamson BOB.WILLIAMSON@ANU.EDU.AU
Australian National University and NICTA
Canberra, ACT 0200, Australia

Editor: David Cohn

Abstract

Open source tools have recently reached a level of maturity which makes them suitable for building large-scale real-world systems. At the same time, the field of machine learning has developed a large body of powerful learning algorithms for diverse applications. However, the true potential of these methods is not used, since existing implementations are not openly shared, resulting in software with low usability, and weak interoperability. We argue that this situation can be significantly improved by increasing incentives for researchers to publish their software under an open source model. Additionally, we outline the problems authors are faced with when trying to publish algorithmic implementations of machine learning methods. We believe that a resource of peer reviewed software accompanied by short articles would be highly valuable to both the machine learning and the general scientific community.

Keywords: machine learning, open source, reproducibility, creditability, algorithms, software

1. Introduction

The field of machine learning has been growing rapidly, producing a wide variety of learning algorithms for different applications. The ultimate value of those algorithms is to a great extent judged by their success in solving real-world problems. Therefore, algorithm replication and application to new tasks are crucial to the progress of the field.

However, few machine learning researchers currently publish the software and/or source code associated with their papers (Thimbleby, 2003). This contrasts for instance with the practices of the bioinformatics community, where open source software has been the foundation of further research (Strajich and Lapp, 2006). The lack of openly available algorithm implementations is a major obstacle to scientific progress in and beyond our community.

We believe that open source sharing of machine learning software can play a very important role in removing that obstacle. The open source model has many advantages which will lead to better reproducibility of experimental results: quicker detection of errors, innovative applications, and faster adoption of machine learning methods in other disciplines and in industry. However, incentives for polishing and publishing software are at present lacking. Published software *per se* does not have a standard, accepted means of citation in our field, and is thus invisible with respect to impact measurement tools like citation statistics: at present the only way of referring to it is by citing the paper which describes the *theory* associated with the code or alternatively by citing the user's manual which has been released in the form of some technical report, such as Benson et al. (2004). To address this difficulty, we propose a method for formal publication of machine learning software, similar to what the ACM Transactions on Mathematical Software provide for Numerical Analysis.

This paper is structured as follows: First, we briefly explain the idea behind open source software (Section 2). A widespread adoption of this publication model would have several positive effects which we outline in Section 3. Next, we discuss current obstacles, and propose possible changes in order to improve this situation (Section 4). Finally, we propose a new, separate, ongoing track for machine learning open source software in JMLR (JMLR-MLOSS) in Section 5. We provide an overview about open source licenses in Appendix A and guidelines for good machine learning software in Appendix B.

2. Open Source and Science

If I have seen further it is by standing on the shoulders of giants.

—Sir Isaac Newton (1642–1727)

The basic idea of open source software is very simple; programmers or users can read, modify and redistribute the source code of a piece of software (Gacek and Arief, 2004). While there are various licenses of open source software (cf. Appendix A; Lin et al., 2006; Välimäki, 2005) they all share a common ideal, which is to allow free exchange and use of information. The open source model replaces central control with collaborative networks of contributors. Every contributor can build on the work that has been done by others in the network, thus minimizing time spent "reinventing the wheel".

The Open Source Initiative (OSI)[1] defines open source software as work that satisfies the criteria spelled out in Table 1. These goals are very similar to the way research works (Bezroukov, 1999):

1. OSI can be found at `http://www.opensource.org`.

1.	Free redistribution
2.	Source code
3.	Derived works
4.	Integrity of the author's source code
5.	No discrimination against persons or groups
6.	No discrimination against fields of endeavor
7.	Distribution of license
8.	License must not be specific to a product
9.	License must not restrict other software
10.	License must be technology-neutral

Table 1: Attributes of Open Source Software from the Open Source Initative

researchers build upon work of other researchers to develop new methods, apply them to produce new results, and publish all of this work, always citing relevant previous work. It is well documented how the move to an "open science" or "open source" model in the Age of Enlightenment (Schaffner, 1994) greatly increased the efficiency of the experimental scientific method (Kronick, 1962) and opened the way for the significant economic growth of the Industrial Revolution (Mokyr, 2005).

However, scientific publications are also not as free as one may think. Major journals are not freely available to the general public since publishers limit access only to subscribers. A few pioneering journals such as the Journal of Machine Learning Research, the Journal of Artificial Intelligence Research, or the Public Library of Science Journals have begun publishing in the so called "open access" model.[2] Open-access literature is digital, online, free of charge, and free of most copyright and licensing restrictions. This model is enabled by low-cost distribution on the Internet, which was economically impossible in the age of print. The "journal pricing crisis" in which journal subscription fees have risen four times faster than inflation since 1986, strongly motivated the development of open access. In summary, open access (with certain limitations) removes *price barriers*, for instance, subscription and licensing fees, and *permission barriers*, that is, most copyright and licensing restrictions. An extensive overview and a time-line concerning this distribution model which our brief summary is also based on, is available from the SPARC Open Access Newsletter.[3] An open letter to the U.S. Congress, signed by 25 Nobel laureates, puts it succinctly:

Open access truly expands shared knowledge across scientific fields, it is the best path for accelerating multi-disciplinary breakthroughs in research.[4]

—Open letter to the U.S. Congress, signed by 25 Nobel laureates, (August 26, 2004)

It is plausible that a similar boost could be expected from a more widespread adoption of open source publication practices in the machine learning field, in which the software implementing the methods would play a comparable role to the underlying theory in the advancement of science. To achieve this, the supporting *software and data* should be distributed under a suitable open source license along with the scientific paper. This is already common practice in some biomedical research, where protocols and biological samples are frequently made publicly available. In the area

2. A list of open access journals is currently maintained at `http://www.doaj.org`.
3. The newsletter can be obtained from `http://www.earlham.edu/~peters/fos/`.
4. The letter is available from `http://www.public-domain.org/?q=node/60`.

of machine learning, this is still rarely the case. However, some freely available benchmark data sets exist, for example, the UCI Repository,[5] the Delve repository,[6] the Caltech 101 data set[7] or Rätsch et al. (2001). Nonetheless, this small number of data sets has had a significant influence on the progress in machine learning, since challenging (in their size or complexity) data collections have helped to calibrate algorithms and to establish their relative merits. For instance, much of the progress of the pattern recognition group at AT&T was tracked in terms of the performance of their algorithms on the NIST and USPS data sets.

In Section 4, we will discuss possible reasons for the current situation in more depth. In the rest of this section, we would like to clarify the notion of "open source" by addressing a common misconception that opening the source makes commercial exploitation impossible. On the contrary, open source software has created numerous new opportunities for businesses (Riehle, 2007). Also, simply *using* an open source program on a day to day basis has little legal implications for a user provided they comply with the terms of their license. Users are free to copy and distribute the software as is. Most issues arise when users, playing the role of a developer, modify the software or incorporate it in their own programs and *distribute a modified product*.

A variety of open source licenses exists, which protect different aspects of the software with benefits for the initial developer or for developers creating derived work (Laurent, 2004). Therefore, there is some flexibility in choosing the license according to the specific needs of the developer, or employer. In the following we give suggestions on which license to choose for common scenarios. This oversimplified description is targeted at developers who just want to "get the program out there".

1. A developer who wants to give away the source code in exchange for *proper credit* for derivative works, even closed-source ones, could choose the *BSD license*. A typical example for this kind of developer would be a researcher who just wants to make his work available to the public, but does not want to prevent inclusion into closed-source software, and also does not rely on getting improvement back from the community. An example for a project using the BSD license is FreeBSD, on which Apple's operating system Mac OS X is partially based.

2. A developer who wants to give away the source code, is comfortable with his source being incorporated into a closed-source product but still wants to *receive bug-fixes and changes* that are necessary *to his source* when integrating the code could choose the *GNU Lesser General Public License (LGPL)*. This developer could be someone who wants to keep developing his software, and by publishing his software basically invites the community to contribute to the software. Using the software as-is in closed-source products is allowed. An example project using this license is the GNU C library, used by nearly all programs on a linux system.

3. A developer who wants to give away the source code and *make sure that his program stays open source*, that is, any extension (or integration) will require both the original and the derived code to be released as open source, could choose the *GNU General Public License (GPL)*. Here, the developer could be a researcher who has further plans with his software and wants to make sure that no closed-source product, not even one of his own *if it includes changes of external developers*, is benefiting from his software. An example of this is the GNU/Linux project.

5. This database is located at `http://mlearn.ics.uci.edu/MLRepository.html`.
6. The website is at `http://www.cs.toronto.edu/~delve/`.
7. The data set is available at `http://www.vision.caltech.edu/Image_Datasets/Caltech101/Caltech101.html`.

License	Apache	BSD/MIT	GPL	LGPL	MPL/CDDL	CPL/EPL
Closed source	Yes	Yes	No	Maybe	Yes	Yes
Commercial	Yes	Yes	No	Maybe	Yes	Yes
Modification release	No	No	Yes	Yes	Yes	Yes
Patent	Yes	No	No	No	Yes	Yes
Jurisdiction	Silent	Silent	Silent	Silent	California	New York
Freedom	PR	Free	PR	PR	Free	PR

Table 2: The rights of the developer to redistribute a modified product. A comparison of open source software licenses listed as "with strong communities" on `http://opensource.org/licenses/category`. The main questions are: whether code can be used in closed source projects (Closed source); whether a program that incorporates the code can be sold commercially (Commercial) without releasing the incorporating program under the same license; whether the source code to modifications must be released (Modification release); whether it provides an explicit license of patents covering the code (Patent); the legal jurisdiction the license falls under (Jurisdiction); freedom to adapt licence terms (Freedom) (PR = Permission Required from license drafter). Apache: License used by the Apache web server; BSD: License under which the BSD Unix variant is released; MIT: developed by the MIT; GPL/LGPL: (lesser) GNU General Public License; MPL: License used by the Mozilla web browser; CDDL: Common Development and Distribution License developed by Sun Microsystems based on the MPL; CPL: Common Public License published by IBM; EPL: Eclipse Public License used by the Eclipse Foundation, derived from the CPL.

All of the open source licenses allow for derivative works (item two in Table 1). In addition it is not possible to limit an open source product to a particular use, for example, to non-commercial or academic use, as it conflicts with item six in Table 1. In a brief summary of common open source licenses, Table 2 shows the rights of a developer to distribute a modified product. A more in-depth discussion about licenses can be found in Appendix A. For more details and a comparison of the various freedoms different licenses provide, see Lin et al. (2006).

Finally, note that the idea of "open source" is not limited to scientific publications and computer software. Authors of other creative works may also want to openly distribute their work. This has created a demand for "open source" type licenses applicable to other media, such as music or images. One of the most prominent movements addressing this demand are the Creative Commons (CC) licenses.[8] The CC project was started in 2001 to supply the analog to open source for less technical forms of expression (Coates, 2007) and extends to all kinds of media like text documents, photographs, video and music. All CC licenses allow copying, distribution, and public performance and display of the work without any license payments. However, CC common terms state that the licenses do not interfere with fair use rights (such as citations, private use etc.), first sale or the freedom of expression and it may restrict the use to, for instance, non-commercial purposes or that no derivative works are allowed (Lin et al., 2006; Välimäki, 2005). It therefore conflicts with the non-discrimination provision in the open source definition (Table 1). It should also be noted that in

8. The creative commons homepage is `http://creativecommons.org/`.

principle anyone can submit a new license to the Open Source Initiative to be certified to comply with the Open Source Definition. Creative Commons does not have such a process but was designed *top-down* (Välimäki, 2005). Applied to the area of science, Creative Commons advocates not only having open source methods, but also open source data and results. It should be noted that open access journals like PLoS use a CC license, namely the Creative Commons Attribution License.[9] The European Union supports a related project towards free exchange of scientific results and data sets.[10]

3. Open Source in Machine Learning

This section of the paper aims to provide a brief overview of open source software and its relationship to scientific activity, specifically machine learning. The reader may think that we are overly positive about the benefits of open source, and do not discuss negative views. The truth is that it is extremely difficult to obtain hard evidence on the debate between proprietary systems and open source software.[11] We argue from moral, ethical and social grounds that open source should be the preferred software publication option for machine learning research and refer the reader to the many advantages of the open source software development (Raymond, 2000). There are also a multitude of advantages of sharing of data and resources, as promulgated in the open science approach (Nature, 2005). Here, we focus on the specific advantages of open source software for machine learning research, which combines the needs and requirements both of being a scientific endeavor, as well as being a producer and consumer of software. They can be categorized into:

1. reproducibility of scientific results and fair comparison of algorithms;
2. uncovering problems;
3. building on existing resources (rather than re-implementing them);
4. access to scientific tools without cease;
5. combination of advances;
6. faster adoption of methods in different disciplines and in industry; and
7. collaborative emergence of standards.

We discuss these points in the following seven subsections.

3.1 Reproducibility and Fair Comparison of Methods

Reproducibility of experimental results is a cornerstone of science. In many areas of science it is only when an experiment has been corroborated independently by another group of researchers that it is generally accepted by the scientific community. It is often the case that experiments are quite hard to reproduce exactly, and in many fields (e.g., medicine) people go to great lengths to try to

9. See for example `http://www.plos.org/oa/definition.html`.
10. The Digital Repository Infrastructure Vision for European Research located at `http://www.driver-repository.eu`.
11. See Section 1.2 of `http://www.dwheeler.com/oss_fs_why.html`.

ensure this. Reproducibility would be quite easy to achieve in machine learning simply by sharing the full code used for experiments.

In the field of machine learning, numerical simulations are often used to provide experimental validation and comparison of methods. Ideally, such a comparison between methods would be based on a rigorous theoretical analysis. For various reasons however, it may not be possible to theoretically analyze a particular machine learning algorithm or to analytically compute its performance in contrast to another. As many methods seek to do well on some real-world problems where the underlying (true) model is unknown, it is very difficult to measure performance in any other way than empirically. In that sense, experiments play a different role than in the natural sciences, as for example physics or chemistry, where experiments are used to better understand certain aspects of nature, instead of algorithms constructed by humans. Nevertheless, the results of the experimental validations are equally important, as these may for instance provide the evidence that a method outperforms existing approaches (or not). Unfortunately, the current practice in the machine learning community is extremely sloppy, as papers get accepted, which are not detailed enough to allow replication.[12] In the pre-internet era, one could perhaps have argued, that for complex algorithms typically used in machine learning, describing every detail would be too lengthy for publication; but nowadays, there would seem to be no such constraints, as supplementary material could be made available online. Indeed, for many complex algorithms one can probably argue, that a clear and well documented program is perhaps the most convenient way of documenting the full details of a machine learning algorithm. So, it follows that an open source approach would be ideally suited to this challenge.

A survey[13] asking JMLR authors for the availability of the system they described in their JMLR papers concluded that about a third specifically said their systems were unavailable for the reasons discussed in Section 4.

My informal survey suggests some authors have a relaxed regard for scientific virtues: reproducibility, testability, and availability of data, methods and programs—the openness and attention to detail that supports other researchers. It's a widespread problem in computer science generally. I'm guilty, too. We programmers tend not to keep the equivalent of lab books, and reconstructing what we have done is often unnecessarily hard. As I wrote elsewhere (see Thimbleby, 2003) there can be problems with publishing work that is not rigorously supported. It is the computer science equivalent of fudging experimental data—whether this really matters for the progress of science is another question.

—Harold Thimbleby, 2003

Reproducing numerical results in order to compare methods is not trivial, as it is often not possible to re-implement a method based only on the information contained in publications. Methods often have a number of free parameters whose correct adjustment requires extensive experience with the specific algorithm, data set, or both. In this context it should be noted that all steps involved in data pre-processing are equally crucial in reproducing results.

The non-reproducibility of results is not merely a theoretical possibility. Consider the recent exchange of papers in this journal (Loosli and Canu, 2007; Tsang and Kwok, 2007). A comment has been published in which the authors document that they could not reproduce the results of another paper. The authors of the original paper defended their original results, blaming the differences

12. One may indeed go further, and ask whether such a practice lives up to the basic requirements of scientific work.
13. A summary is at `http://www.uclic.ucl.ac.uk/harold/srf/jmlr.html`.

on the operating system used to perform the experiments. Needless to say, such a situation is unsatisfactory. This example also reflects another benefit of making source code available: it allows us to uncover hidden tricks that remain typically undocumented (Orr and Müller, 1998). The reason a certain implementation of a machine learning method outperforms all other approaches with similar algorithms may be due to a number of functions that have been tuned to specific machine instructions.

Furthermore, instances of fraud or scientific misconduct can be more easily detected if all the code required to perform the experiment is made available. Thus, making algorithms, *including* the source code and data publicly available (such as the efforts mentioned in Section 2) significantly enhances the reproducibility and the feasibility of (fair) comparisons.

3.2 Quicker Detection and Correction of Bugs

An important feature that has contributed much to the success of open source software is that with the availability of the source code, it is much easier to spot and fix bugs in software. While not everyone would be inclined (or able) to satisfactorily resolve a bug himself, everybody has the possibility to inspect the source code, find the bug and submit a patch to the maintainers of the project. This observation has been summarized as "Given enough eyeballs, all bugs are shallow", known as Linus's Law (Raymond, 2000). Further, to paraphrase Al Viro,[14] all software contains bugs, be it open-source or proprietary. The only question is what can be done about a particular instance of software failure, and that is where having the source matters.

3.3 Faster Scientific Progress by Reduced Cost for Re-implementation of Methods

Scientific progress always builds on existing publications and methods. The field of machine learning is no exception. However, re-implementing existing methods in order to test them, use them as part of a larger project, or to extend them, is a large burden on the researcher. This is particularly true for method oriented research. As already discussed above, publications often do not contain all the information necessary to re-implement a method. The complexity of existing methods is often so large that re-implementing its algorithms can require prohibitive effort.

As a consequence, work on such methods is often restricted to a few groups who already have implementations, and newcomers to the field have to first redo the work of others. Alternatively, such a situation can lead to ignoring existing competitors since implementations are not available, and re-implementation seems infeasible. Therefore, the availability of open source implementations can help speed up scientific progress significantly.

3.4 Long Term Availability and Support

For the individual researcher, open source may provide a means of ensuring that he will be able to use his research even after changing his employer. Even the most generous institutions tend to introduce delays before giving *formal* approval for code reuse after the researcher moves. This is, however, *harmful for both researcher and employer*: obviously for the researcher since he loses access to the tools he has been working with but also for the institution since the piece of code

14. This quotation can be obtained from the linux kernel mailinglist `http://www.ussg.iu.edu/hypermail/linux/kernel/0404.3/1344.html`.

in question becomes *unsupported*. By releasing code under an open source license the chances of having long-term support are dramatically increased.

3.5 Combination of Advances

Scientific progress does not always occur as paradigm shifts (e.g., the emergence of Decision Trees, Neural Networks, Kernel Methods, Boosting, and Graphical Models) but it is much more likely to occur by incremental improvements over a given existing technique. Moreover, it is likely that several such changes occur simultaneously once a given topic reaches the mainstream. While this is, in principle, a good thing, it poses a rather unique problem: how to combine several of those advances into *one* joint implementation.

As a case in point, consider progress in kernel methods. There is currently no piece of code or even a publication which combines structured estimation, semiparametric methods, automatic margin adjustment, different types of regularization, methods for dealing with missing variables, methods for dealing with invariances, a large set of kernel functions, nonconvex approximations of the loss, leave-one-out estimators, or transductive estimation. While each of these modifications are well established and it is commonly accepted that they work, there is no publication indicating the performance of a combination of more than three of the ten aforementioned methods.

This is more than just a simple nuisance: it is not clear at all whether the combination of all of those "improvements" would really be beneficial and what their interactions might be. Do some of these methods effectively solve the same problem and derive their gains from a common change in the estimate? What are the computational limitations?

Without access to a common codebase *and* willingness of the community to improve upon it it will be next to impossible to address this issue, since it is likely to be too difficult for a single researcher to track and compare all modifications.

3.6 Faster Adoption in Machine Learning, Other Disciplines and Industry

Availability of high-quality open source implementations can ease adoption by other machine learning researchers, users in other disciplines and developers in industry for the following reasons:

1. Open source software can be used without cost in teaching.

2. If a method proves useful and its source code is available, it can be directly applied to related real world problems in other fields or in industry.

In areas such as bioinformatics, the expertise to implement advanced machine learning methods from scratch is often not available. While this situation might be perceived as desirable by some to ensure that machine learning experts are sought by the industry, hiring machine learning experts will become more desirable for companies as the field gains prominence. In fact, one may argue that it is the problem of *automatic* adjustment and deployment that machine learning theory should be addressing by suitable means of model selection. Having access to an extensive machine learning toolkit will allow us to compare model selection techniques in realistic settings.[15] Increased distribution of machine learning's end-product, software, will lead to more success stories of its use within industrial applications. Publishing software as open source might also be the

15. See, for example, the NIPS'04 workshop on the (Ab)Use of Bounds `http://www.hunch.net/~jl/conferences/abuse_of_bounds/abuse_of_bounds.html`.

only means to reach wide-spread distribution of your software if you lack the logistic infrastructure of big companies like Microsoft™. In addition, the adoption of machine learning methods in large-scale applications can have a very stimulating effect on the field itself, and lead to novel and interesting challenges. It still requires an expert with deep understanding of the method to adjust it to a particular application. There are also impressive precedents of open source software leading to the creation of multi-billion dollar companies and industries.[16]

3.7 Collaborative Moves towards Better Interoperability

The diversity of machine learning forbids a single, mono-cultural software framework satisfying all needs. However, even in areas where it is in principle feasible, most pieces of machine learning software do not inter-operate very well, because of differences in interfaces, data abstractions and work flows. Ultimately it would be desirable to agree to a set of standards which ensure, for example, that data sets can be exchanged between machine learning tools, and that classification algorithms can be interchanged seamlessly.

However, given the distributed nature of scientific work, it is unlikely that a centralized institution can be formed which develops such standards in a top-down manner. Now with the publication of toolboxes according to an open source model, it becomes possible for individual projects to move towards standardization in a collaborative, distributed manner.

This process has already begun, mostly with toolboxes incorporating other toolboxes or providing "glue" code to access functionality contained in other toolboxes. A typical example are the libraries for learning support vector machines, such as LIBSVM (Chang and Lin, 2001), SVMLin (Sindhwani and Keerthi, 2006), SVMTorch (Collobert and Bengio, 2001) and GPDT (Zanni et al., 2006). A small sample of larger frameworks which provide access to (among other features) one or more of these libraries include Elefant (Gawande et al., 2007), Orange (Demsar and Zupan, 2004), PLearn (Vincent et al.), RapidMiner[17], Shogun (Sonnenburg et al., 2006), Torch (Collobert et al., 2002) and the Weka (Witten and Frank, 2005) toolboxes.

In the future, instead of the constant repetition of work, standards should emerge, pushed either by library and/or toolbox developers, in order to make this integration much less difficult. A consensus could also emerge via dialog in journal or community websites.[18] Which standards will be adopted will depend on the popularity of the individual toolboxes or libraries.

We conclude this section by summarizing the advantages described above in Table 3.

4. Current Obstacles to an Open Source Community

While there exist many advantages to publishing implementations according to the open source model, this option is currently not taken often. We believe that there are six main reasons which will be discussed in greater detail in the next sections.

---

16. Perhaps the oldest, dating from the early 1970s, is SPICE (Simulation Program with Integrated Circuit Emphasis) (Wikipedia, 2007b), which has led to the foundation of Synopsys and Cadence Design Systems, and significantly grew the whole Electronic Design Automation Industry.
17. Former YALE toolbox, available from `http://www.rapidminer.com`.
18. We propose to use `http://mloss.org` as *the* platform for machine learning open source software (MLOSS) to openly discuss design decisions and to host and announce MLOSS.

1.	Reproducibility of scientific research is increased
2.	Algorithms implemented in same framework facilitate fair comparisons
3.	Problems can be uncovered much faster
4.	Bug fixes and extensions from external sources
5.	Methods are more quickly adopted by others
6.	Efficient algorithms become available
7.	Leverage existing resources to aid new research
8.	Wider use leads to wider recognition
9.	More complex machine learning algorithms can be developed
10.	Accelerates research
11.	Benefits newcomers and smaller research groups

Table 3: Eleven Advantages of Machine Learning Open Source Software

4.1 Publishing Software is Not Considered a Scientific Contribution

Some researchers may not consider the extra effort to create a usable piece of software out of machine learning methods to be science. However, machine learning is a synthetic discipline as well as an analytic one, and certainly if it is science it is in Simon's phrase, a "Science of the Artificial" (Simon, 1969), in which artifacts, specifically implemented algorithms, is one of the major outputs. In addition to the "pure" scientific pursuits, machine learning researchers also produce technological outputs. As such, the discipline could be considered to be mathematical engineering. In any case, as was pointed out in Section 3, the complexity of existing methods is growing such that re-implementing algorithms can easily take months. Some argue that if you want to really understand an algorithm and want to extend it—which is an important task for machine learning researchers—you have to implement it from scratch and thus it is not beneficial to have the software available. This is only partially true: one does not want to reimplement all the basic algorithms an advanced method builds on, but simply understand the high-level steps. After all, one has to build upon existing libraries, as for example the standard or math library, the Basic Linear Algebra Subprograms (BLAS) (Lawson et al., 1979), the Linear Algebra PACKage (LAPACK) (Anderson et al., 1999) to be productive. Only few people would want to re-implement, or would be able to generate a high quality implementation of, common sorting algorithms such as `qsort`, basic mathematical functions such as `sin`, or linear algebra operations such as `dgemm` or `dgesv`.

4.2 Misconception—Opening the Source Conflicts with Commercial Interests

As already discussed in Section 2, there is a common misconception that opening the source makes commercial use—licensing of commercial versions or use in industrial projects—impossible. It may, however, prevent the creation of closed-source products that include external open-source contributions. In reality, careful selection of a suitable open source license would satisfy the requirements of most researchers *and their employer*. For example, using the concept of *dual licensing* one could release the source code to the public under a open source license with strong reciprocal obligations (like the GNU GPL), and at the same time sell it commercially in a closed-source product. In Appendix A we give a few hints for choosing an appropriate license.

4.3 The Incentive for Publishing Open Source Software is not High Enough

Unlike writing a journal article, releasing a piece of software is only the beginning. Maintaining a software package, fixing bugs or writing further documentation requires time and commitment from the developer, and this contribution is also rarely acknowledged. Open source programmers often gain a good "reputation" among their peers, which in some situations may be worth more than citations (Kelty, 2001; Franck, 1999). But scientific success, especially in research institutions, is often determined by measures such as citation statistics. However, there exists no academic, widely accepted platform to publish software. As a result, researchers tend to not acknowledge software used in their published research, and the effort which has to be expended to turn a piece of code for personal research into a software product that can be used, understood, and extended by others is not sufficiently acknowledged. As just one example, a well-known structured classification method had 766 Google Scholar citations as of this writing, while the supporting software, which was released with an open-source license but no peer-reviewed publication, has only 78 citations. In contrast, published software descriptions for bioinformatics programs are cited in every published use of the program: the published description of one version of BLAST had 20540 Google Scholar citations, for instance.

4.4 Machine Learning Researchers are Not Good Programmers

While most machine-learning methods are implemented in some form, it does not follow that the best machine learning researchers are the best programmers. Opening up "research quality" code to the inspection and modification of others (who may be more skilled programmers) can certainly help to improve the quality of the code base. On the other hand, the initial developers may be reluctant to expose their programming practices to public criticism.

4.5 Sloppiness Hides Problems of Newly Proposed Methods and Eases Acceptance at Conferences and Journals.

A certain degree of sloppiness may be advantageous to someone trying to promote a new method. For example, many algorithms require the setting of parameters, decisions about convergence, and a multitude of other things, and it is perhaps not unusual that researchers inadvertently "help their new algorithms along", by carefully making sure that "nothing goes wrong" during the application of a method, and if something does go wrong, a suitable measure is taken, that is, reduction of a learning rate, restart with a new random seed etc. Thus, being absolutely precise about the algorithm, could help bring these issues to the surface, but this is currently only rarely done, presumably because such details are thought of as secondary, and not really part of the idea of the algorithm. Therefore, at first glance, making the source code for a particular machine learning paper public may seem counter-productive for the researcher, as other researchers can more easily find problems with the proposed method, and possibly even discredit the approach. The researcher may also lose a competitive advantage because competing groups can also use the software. However, the same argument holds for making research papers publicly available, and as discussed in Section 2 the move to an open science in the Age of Enlightenment sped up scientific progress and boosted economic growth. Therefore, the already altruistic behavior of publishing papers should be complemented by also providing open source code as the same great benefits can be expected if many other researchers follow this path and also distribute accompanying open source software.

4.6 Tradition—Reviewers Pass Papers of Similar Quality

Finally, there seems to exist a tradition, which let's people "get away" with less. When reviewers examine a paper, they have other similar papers (they passed) in mind. They therefore pass papers "for tradition", although the papers could have become a lot more valuable, if reviewers required that the source code of the algorithm had been provided.

These latter two issues are closely related to the question of how to design experiments in a way which ensures the ability to make strong statistical claims about the outcomes of experiments. One such attempt was made in the DELVE (Data for Evaluating Learning in Valid Experiments) archive. However, this archive never gained much popularity, presumably because its data sets are typically not very large, and it has proven to be difficult to reach statistically strong conclusions using relatively small data sets.

5. Proposal

In summary, providing open source code would help the whole community in accelerating research. Arguably, the best way to build an open source community of scientists in machine learning is to promote open source software through the existing reward system based on citation of archival sources (journals, conferences). Unfortunately, persuading people to publish the implementation together with their research paper is a long-term process, exacerbated by a potentially conflicting industrial interest. However, it is possible that a push in this direction could gather momentum, with peer pressure doing the rest.

We would like to initiate this process by giving researchers the opportunity to publish their machine learning open source software, thereby setting an example of how to deal with this kind of publication media. The proposed new JMLR track on machine learning open source software with review guidelines specially tailored to the needs of software is designed to serve that purpose.

We encourage submissions which are contributions related to implementations of non-trivial machine learning algorithms, toolboxes or even languages for scientific computing. As with the main JMLR papers, all published papers will be freely available online. The software must adhere to a recognized open source license (`http://www.opensource.org/licenses/`). Submissions should clearly indicate that they are intended for this special track in the cover letter of the submission.

Since we specifically want to honor the effort of turning a method into a highly usable piece of software, prior publication of the method is admissible, as long as the software has not been published elsewhere. As an inspiration we discuss in Appendix B basic software design principles and more machine learning (toolbox) related ideas. In summary, preparing research software for publication is a significant extra effort which should also be rewarded as such.

It is hoped that the open source track will motivate the machine learning community towards open science, where open access publishing, open data standards and open source software foster research progress.

5.1 Format

We invite submissions of descriptions of high quality machine learning open source software implementations. Submissions should at least include:

1. A cover letter stating that the submission is intended for the machine learning open source software section, the open source license the software is released under, the web address of the project, and the software version to be reviewed.

2. An up to four page description based on the JMLR format.

3. A zip or compressed tar-archive file containing the source code and documentation.

5.2 Review Criteria

The following guidelines would be used to review submissions. While ideally submissions should satisfy all the criteria below, they are not necessary requirements. Some examples of acceptable submissions which do not satisfy all criteria are: well designed open source toolboxes based on Matlab^{TM}; learning algorithms using commercial optimizers such as MOSEK or CPLEX as a back-end; or a teaching tool which has poor computational performance due to its didactic implementation.

1. The quality of the four page description.

2. The novelty and breadth of the contribution.

3. The clarity of design.

4. The freedom of the code (lack of dependence on proprietary software).

5. The breadth of platforms it can be used on (should include an open-source operating system).

6. The quality of the user documentation (should enable new users to quickly apply the software to other problems, including a tutorial and several non-trivial examples of how the software can be used).

7. The quality of the developer documentation (should enable easy modification and extension of the software, provide an API reference, provide unit testing routines).

8. The quality of comparison to previous (if any) related implementations, w.r.t. run-time, memory requirements, features, to explain that significant progress has been made.

After acceptance, the abstract including the link to the software project website, the four page description and the reviewed version of the software will be published on the JMLR-MLOSS website `http://www.jmlr.org/papers/mloss`. The authors can then make sure that the software is appropriately maintained and that the link to the project website is kept up-to-date.

6. Conclusion

We have argued that the adoption of the open source model of sharing information for implementations of machine learning software can be highly beneficial for the whole field. The open source model has many advantages, such as improved reproducibility of experimental results, quicker detection of errors, accelerated scientific progress, and faster adoption of machine learning methods in other disciplines and in the industry. As the incentives for publishing open source software are currently insufficient, we outlined a platform for publishing software for machine learning. Additionally, we discussed desirable features of machine learning software which will ultimately lead to highly usable, flexible and scalable software. We invite all machine learning researchers developing machine learning algorithms to submit to the new JMLR track for machine learning software. Defining well-designed interfaces will prove crucial towards better interoperability, leading to a community built suite of high-quality machine learning software.

Researchers in machine learning should not be content with writing small pieces of software for personal use. If machine learning is to solve real scientific and technological problems, the community needs to build on each others' open source software tools. Hence, we believe that there is an urgent need for machine learning open source software. Such software will fulfill several concurrent roles: a better means for reproducing results; a mechanism for providing academic recognition for quality software implementations; and acceleration of the research process by allowing the standing on shoulders of others (not necessarily giants!).

Acknowledgments

The authors would like to acknowledge S.V.N. Vishwanathan, Torsten Werner and the attendees of the NIPS Workshop on Machine Learning Open Source Software 2006 for inspiring discussions. We thank Andre Noll and Sebastian Schultheiß whose careful reading and insights have improved this manuscript. We thank Evana Ushakoff for providing legal comments. We gratefully acknowledge partial support from the PASCAL Network of Excellence (EU #506778). C. S. Ong is also with Max Planck Institute for Biological Cybernetics, Spemannstr. 38, 72076 Tübingen, Germany and K. R. Müller with the Fraunhofer Institute FIRST, Kekulestr. 7, 12489 Berlin, Germany.

Appendix A. Which License to Choose?

As discussed in Section 2, most issues regarding the use of open source software arise when one wants to distribute a modified or derived product. In this section, we wish to discuss these issues in more depth.

With the proliferation of open source software, various licenses have been put forward, confusing a developer who just wants to release his program to the public. Whilst the choice of license might be considered a boring legal/management detail, it is actually very important to get it right—the choice of certain licenses may significantly limit the impact a piece of software may have.[19] In this section we briefly summarize some pertinent questions below as a guideline to some of the more popular licenses.[20]

The owner of the intellectual property present in the code (often the original author, but depending on employment contract, sometimes the employer) owns the copyright of the work and can

19. For example if the SPICE software had been released under a GPL-like license, it is extremely unlikely that it would have had the impact that it did, with multi-billion dollar companies being created on the basis of it because the value-add the companies created could not have been protected, and thus there would be no competitive advantage. On the other hand it is questionable whether the Linux kernel would have evolved into an open, full featured multi-platform kernel with thousands of developers continuously contributing if it was BSD licensed.
20. Disclaimer: This does not constitute legal advice. Since licensing *is* a legal issue, and since employers usually have an interest in the protection of what is usually their intellectual property, readers should always seek their own formal legal advice.

thus dictate the license under which it is released (Webbink, 2003). Different licenses protect different aspects of the software with benefits for the initial developer or developers creating derived work (Laurent, 2004). Significant licensing issues may arise when open source software (OSS) is combined with proprietary code. Depending on the license, the resulting product may have to be published as open source, including the formerly proprietary code. Licenses which demand that subsequent modifications of the software be released under the same license are called "copyleft" licenses Wikipedia (2007a), the most famous of which is the GNU General Public License (GPL).

For example, for developers creating derived works a BSD/MIT license is the most liberal, as it allows a developer to incorporate the software in his own product, without open sourcing the whole product later; and GPL is the most strict, trying to ensure that all subsequent derivatives of the software also stay open. From the viewpoint of the original developer, this situation is reversed: Using the BSD/MIT license, he may not benefit from patches with enhancements, while using the GPL license ensures that derived work will stay open, making future enhancements available to the original developer. Then there are the "in between" licenses, like Lesser GNU General Public

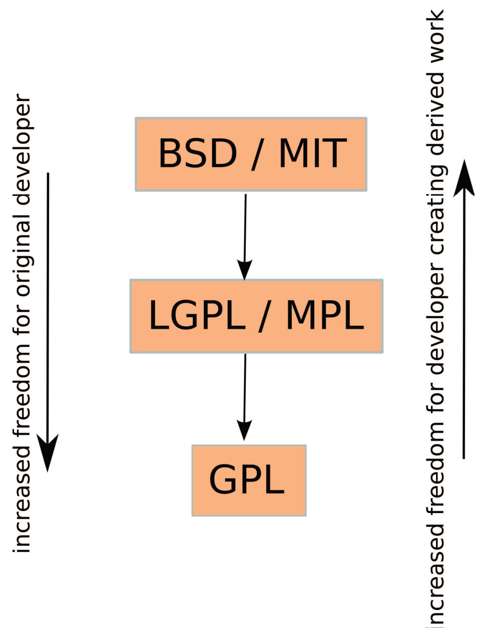

Figure 1: An illustration of open source licenses with respect to the rights for the initial developer and the developer creating derived works.

License (LGPL), the Common Public License (CPL) and the Mozilla Public License (MPL) that only require the changes to the code to be released. Hence the original author has access to any future modifications (bug fixes or new features) of his or her particular piece of software. Figure 1 visually illustrates license interdependencies.

Note that one can release one's own software under multiple licenses. This is referred to as *dual licensing* and allows a developer to release his code to the public under the GPL and at the same time sell it commercially in a closed-source product. However if one includes changes in a program that other developers have made contributions, the agreement of all contributors is required to change a license (Laurent, 2004; Burnette, 2006; Fitzgerald and Bassett, 2003). A crude summary of some of the simple distinctions between some OSS licenses is given in Table 2. It should be noted that a simple table hides the complexity of some of the key issues (see below).

A.1 Some Complexities

As an illustration of some of the difficulties, let us consider the issue of conflicting open source licenses and the issue of reciprocal obligations.

A.1.1 Open Source Licenses may Conflict

When releasing a program as "open source" it is not obvious that although the program is now "open source" it still may have a license that conflicts with many other open source licenses. Licenses may have mutually conflicting requirements, for example with respect to jurisdiction, or including advertising clauses, such that one cannot legally combine the two programs into a new derived work (simply using both programs is usually possible, though). The OSI currently lists 60 open source licenses[21] and the consequence of this license proliferation[22] means that the simple inclusion BSD $\subset$ LGPL $\subset$ GPL as shown in Figure 1 does not hold for other licenses.[23] For example the MPL and CPL conflict with the most widely used licenses, which are the GPL (in use by about 70% of the OSS programs) and the LGPL (about 10% spread), and may even conflict with each other Figure 2. While this can be used to purposely generate conflicts, as a general rule, one should refrain from doing so as it will make code exchange between open source projects impossible and may limit distribution and thus success of a open source project. For a more in-depth discussion see Wheeler (2007). Researchers aspiring to a wide developer audience for their software should consider GPL compatible licenses,[24] or select one with a strong community.[25]

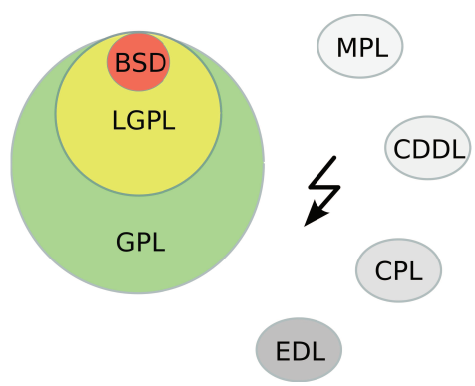

Figure 2: Open Source Licenses may conflict with each other.

21. The OSI license list can be found at `http://www.opensource.org/licenses/alphabetical`.
22. The license proliferation committee report is available at `http://opensource.org/osi3.0/proliferation-report`.
23. Note that this only holds for the 3-clause BSD license. Also note that this is a one-way street, that is, BSD licensed software cannot merge code from LGPL/GPL and LGPL cannot merge software from GPL projects
24. The Free Software Foundations GPL compatible license list is available at `http://www.gnu.org/philosophy/license-list.html`
25. Licenses with a strong community are listed at `http://opensource.org/licenses/category`.

A.2 Reciprocal Obligations

Another issue is the one of reciprocal obligations: any modifications to a piece of open source software may need to be available to the original authors. In the following, to give a hint of the complexity, reciprocal obligations are discussed for the following licenses:

- LGPL – applies the concept of "derivative works", which (confusingly) can include the combined work resulting from linking a LGPL-licensed Library and a non-LGPL "work that uses the Library". Problematically, the LGPL requires for such combined works that the source code of the "work that uses the Library" needs to be disclosed when the combined work is distributed (LGPL section 2, third last paragraph). This is a substantial limitation to the utility of the LGPL in enabling components to be further developed and distributed with proprietary code. The LGPL also tries to make some fairly complex and unclear distinctions between what constitutes a collective or derivative work to determine whether the LGPL attaches to licensee-created works.

- MPL – does not apply the concept of "derivative works", but talks instead of "modifications" to (i.e., additions to or deletions from) the Original Code as comprising part of the Covered Code (i.e., code to which the MPL applies). This makes the MPL more comprehensible to (some) legal audiences, and therefore more certain from that perspective. However, it also makes the MPL's reciprocal obligation more limited. The MPL permits Covered Code to be distributed within Larger Works in a combined work without the MPL attaching to the non-MPL code (as long as the distributor continues to apply the MPL to the Covered Code component of the Larger Work). This overcomes the over-inclusiveness aspect of the GPL and LGPL, and makes the MPL more friendly towards developers who may wish to combine MPL code with their own proprietary code that is not a "modification" of MPL code.

- CPL – like MPL, applies the concept of additions or changes from the original Contribution. However, the CPL arguably imposes more narrow reciprocity obligations than either GPL/LGPL or MPL, because the CPL explicitly exempts the reciprocity obligations from applying to a "separate module of software distributed in conjunction" with the original Contribution that is not a "derivative work". Put another way, the CPL reciprocity obligation only attaches to additions to the original contribution that are "derivative works" but not separate modules of software.

Appendix B. Guidelines for Good Machine Learning Software

Without claiming to be exhaustive, in this appendix we record some guidelines which, we believe, would lead to high quality machine learning software.

B.1 Good Software Practices

There is a significant difference between a piece of code which is intended to be used privately (either alone or within a small research group), and one which is intended to be made public and to be used (or even extended) by external users. While a certain lack of organization, documentation, and robustness can be tolerated when the software is used internally, it can make the software next to useless for others. The old rule that software is primarily written for other humans, and not only for computers, is even more important when your audience is larger than colleagues with whom you closely collaborate.

1.	Software is useful and usable.
2.	Software is documented.
3.	Software is robust.
4.	Software has well-defined interfaces.
5.	Software uses existing interfaces and standards.
6.	Software has well established (unit) testing routines.

Table 4: Six features of useful machine learning software

Good machine learning software should first of all be a good piece of software (Table 4). There exist many books on software design. The inclined reader is referred to the books by Raymond (2004) or Hunt and Thomas (2000) for further information. Just putting your research software on your web-page will not be sufficient. One should follow general rules for developing open source software (see also the discussion by Levesque, 2004, which highlights common failure modes for open source software development):

- The software should be structured well and logically such that its usability is high.

- It should be documented well, such that you can learn to use the software quickly; for example, in the form of a tutorial, a reference, and examples; ideally, also include developer's documentation which explains the software's internals;

- It should be sufficiently robust, which means that it is as much as possible bug-free, but also tolerates incorrect inputs as well as providing meaningful error messages instead of breaking down silently.

- It should provide testing routines to verify automatically whether the code is correct. This reduces the likelihood that modifications of the code introduces bugs.

In any case, the main goal should be to maximize the re-usability of your software. Therefore you would want to make your software as flexible as possible such that it can deal with a large number of different types of data. You would also want to clearly define the interface to your software such that others can easily use it directly.

Ideally, the software also includes a number of unit tests. These are small programs which can be run automatically and test the individual parts of the program for correctness. Unit tests are an indispensable tool for ensuring that a small change does not introduce bugs which go unnoticed for a long time. Such tests therefore facilitate modification of software greatly.

Apart from these considerations that apply to any software design, there are several requirements that are specific to the domain of machine learning. Since these requirements are quite different depending on whether you are writing high-quality implementations of a specific algorithm (for example, support vector machines), or more general frameworks, we have split the discussion in two sections discussing these two extremes.

B.2 Guidelines for Single Machine Learning Algorithms

Consider the case that a researcher has special expertise in implementing a certain class of machine learning algorithms, and has developed, for example, a new implementation of support vector machines which is very efficient, or a clever implementation of a certain class of graphical models. There should exist several ways of using the program, for example, stand-alone from the command line, and as a library which can be linked to other programs. Reusing the interface of existing software solving the same problem is also very useful. Then, the software can be used as a drop-in replacement. If the algorithm can be practically applied to large data sets, it is desirable that the available main memory is not the limiting factor, but if the algorithms are designed such that they can also deal with data sets which reside on the hard disk, using the main memory as a cache. Finally, one should make sure that the software is able to read and write data formats in at least one commonly used data exchange standard.

B.3 Guidelines for Larger Machine Learning Frameworks

A completely different kind of endeavor is to build a framework, or an environment, which can be used for a large number of different machine learning tasks. Such a framework typically integrates a number of existing more specialized machine learning algorithms, or low-level numerical libraries.

Since frameworks should be suited for a wide range of applications—potentially including methods and data types which have not yet been invented—a clean design is particularly important. One approach to achieve this is to decompose the framework into several small modules with clearly defined interfaces so as to minimize the coupling between different parts of the framework. Then, individual modules can be modified or extended more easily.

For example, a framework which deals with vectorial data and matrices, could also provide access to a standard set of basic linear algebra routines, learning algorithms dealing with vectorial data like support vector machines, or least squares regression, and routines to store and read these standard data types. However, the interfaces between these components are sufficiently abstract that it is possible to replace the linear algebra routines by more efficient ones without affecting the rest of the framework.

But as machine learning deals with a large number of different kinds of data sets, frameworks could also support other data types like strings, sequences, trees, graphs, sparse vectors, et cetera. Likewise, tools for graphical models should allow for easy specification of the model, ability to save states, a variety of approximate samplers and solvers, convergence monitors, and flexible nonparametric message passing tools.

Beyond these basic features, the following methods would be nice to have: efficient optimization solvers; access to classical statistical methods and probability distribution; a good visualization library, that provides graphs of various kinds to help analyzing data and reporting results; various classification and regression algorithms, also with extensions to one-class and multi-class; clustering and structure learning algorithms; graphical models, and Bayesian inference, et cetera.

As clusters of machines become more and more affordable, it would be nice to provide simple ways to parallelize parts of the algorithms. Often machine learning algorithms are easy to parallelize and only the barrier of low-level parallel computing stops the designers from doing so. To achieve this goal parallel libraries such as OpenMP and MPI could be used.

References

Ed Anderson, Zhaojun Bai., Christian Bischof, Susan Blackford, James Demmel, Jack Dongarra., Jeremy Du Croz., Anne Greenbaum, Sven Hammarling, Alan McKenney, and Danny Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

Steven J. Benson, Lois Curfman-McInnes, Jorge Moré, and Jason Sarich. TAO user manual (revision 1.8). Technical Report ANL/MCS-TM-242, Mathematics and Computer Science Division, Argonne National Laboratory, 2004. `http://www.mcs.anl.gov/tao`.

Nikolai Bezroukov. Open source software development as a special type of academic research (critique of vulgar Raymondism). *First Monday*, 4(10), October 1999. `http://www.firstmonday.org/issues/issue4_10/bezroukov/index.html`.

Ed Burnette. How to pick an open source license. `http://blogs.zdnet.com/Burnette/?p=130` and `http://blogs.zdnet.com/Burnette/?p=131`, 2006.

Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: A library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Jessica Coates. Creative commons – the next generation: Creative commons licence use five years on. *SCRIPT-ed*, 4(1), 2007. `http://www.law.ed.ac.uk/ahrc/script-ed/vol4-1/coates.asp`.

Ronan Collobert and Samy Bengio. SVMTorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001. ISSN 1533-7928.

Ronan Collobert, Samy Bengio, and J. Mariethoz. Torch: A modular machine learning software library. Technical report, IDIAP, 2002. IDIAP-RR 02-46.

Janez. Demsar and Blaz. Zupan. Orange: From experimental machine learning to interactive data mining, white paper `http://www.ailab.si/orange`. Technical report, Faculty of Computer and Information Science, University of Ljubljana., 2004.

Brian Fitzgerald and Graham Bassett. Legal issues relating to free and open source software. In *Legal Issues Relating to Free and Open Source Software*, pages 11–36. Queensland University of Technology, School of Law, 2003. `http://www.law.qut.edu.au/files/open_source_book.pdf`.

Georg Franck. Scientific communication – a vanity fair? *Science*, 286(5437):53–55, 1999.

Cristina Gacek and Budi Arief. The many meanings of open source. *IEEE Software*, 21(1):34–40, 2004.

Kishor Gawande, Christfried Webers, Alexander J. Smola, and S.V.N. Vishwanathan. ELEFANT: A python machine learning toolbox. In *SciPy Conference*, 2007.

Andrew Hunt and David Thomas. *The Pragmatic Programmer*. Addison-Wesley, 2000.

Christopher M. Kelty. Free software/free science. *First Monday*, 6(12), December 2001. http://www.firstmonday.org/issues/issue6_12/kelty/index.html.

David A. Kronick. *A history of scientific and technical periodicals: the origins and development of the scientific and technological press, 1665-1790*. Scarecrow Press, New York, 1962.

Andrew M. St. Laurent. *Open Source & Free Software Licensing*. O'Reilly Media, Inc, 2004. http://www.oreilly.com/catalog/osfreesoft/book/.

Charles L. Lawson, Richard J. Hanson, David Kincaid, and Fred T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Soft.*, 5:308–323, 1979.

Michelle Levesque. Fundamental issues with open source software development. *First Monday*, 9 (4), April 2004. http://www.firstmonday.org/issues/issue9_4/levesque/index.html.

Yi-Hsuan Lin, Tung-Mei Ko, Tyng-Ruey Chuang, and Kwei-Jay Lin. Open source licenses and the creative commons framework: License selection and comparison. *Journal of Information Science and Engineering*, 22:1–17, 2006.

Gaëlle Loosli and Stéphane Canu. Comments on the "core vector machines: Fast SVM training on very large data sets". *Journal of Machine Learning Research*, 8:291–301, 2007.

Joel Mokyr. The intellectual origins of modern economic growth. *The Journal of Economic History*, 65(2):285–351, 2005.

Nature. Let data speak to data. *Nature*, 438(7068):531, 2005.

Open Source Initative. http://www.opensource.org/docs/osd.

Genevieve B. Orr and Klaus-Robert Müller, editors. *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*. Springer, 1998.

Gunnar Rätsch, Takashi Onoda, and Klaus-Robert Müller. Soft margins for AdaBoost. *Machine Learning*, 42(3):287–320, 2001. Data is hosted at http://ida.first.fraunhofer.de/projects/bench.

Eric S. Raymond. The cathedral & the bazaar. 2000. http://www.tuxedo.org/~esr.

Eric S. Raymond. *The Art of UNIX Programming*. Addison-Wesley, 2004.

Dirk Riehle. The economic motivation of open source software: Stakeholder perspectives. *IEEE Computer*, 40(4):25–32, 2007.

Ann C. Schaffner. The future of scientific journals: Lessons from the past. *Information Technology and Libraries*, 13(4):239–247, 1994.

Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, Massachusetts, first edition, 1969.

Vikas Sindhwani and S. Sathiya. Keerthi. Large scale semi-supervised linear svms. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 477–484, New York, NY, USA, 2006. ACM Press.

Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7:1531–1565, July 2006.

Jason E. Strajich and Hilmar Lapp. Open source tools and toolkits for bioinformatics: significance, and where are we? *Briefings in Bioinformatics*, 7(3):287–296, 2006.

Harold Thimbleby. Explaining code for publication. *Software Practice and Experience*, 33(10): 975–1001, 2003.

Ivor W. Tsang and James T. Kwok. Author's reply to the "comments on the Core Vector Machines: Fast SVM Training on Very Large Data Sets". *Journal of Machine Learning Research*, 2007. submitted.

Mikko Välimäki. *The Rise of Open Source Licensing*. Turre Publishing, 2005.

Pascal Vincent, Yoshua Bengio, and Nicolas Chapados. `http://plearn.org`.

Mark Webbink. Licensing and open source software. In *Legal Issues Relating to Free and Open Source Software*, pages 1–11. Queensland University of Technology, School of Law, 2003. `http://www.law.qut.edu.au/files/open_source_book.pdf`.

David A. Wheeler. Make your open source software GPL-compatible. or else. `http://www.dwheeler.com/essays/gpl-compatible.html`, August 2007.

Wikipedia. Copyleft – Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Copyleft`, 2007a. [Online; accessed 2-July-2007].

Wikipedia. SPICE – Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/SPICE`, 2007b. [Online; accessed 29-June-2007].

Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2005. 2nd Edition.

Luca Zanni, Thomas Serafini, and Gaetano Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7: 1467–1492, July 2006.