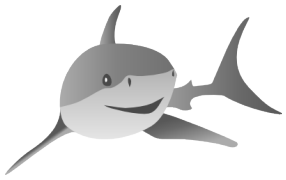
Development Plan for OSS World Challenge 2011

Registration No.	2011- ※Registration No. need not be written.
Program title	**The Shark Machine Learning Library** 

1. Program Overview

SHARK is a fast, modular, state-of-the-art, open-source C++ machine learning library. It is both a rich framework for **academic research** as well as a reliable and well-performing toolbox for **real-world pattern recognition applications**. The Shark library features many **unique algorithms** to our knowledge not available in any other machine learning library. It also covers a broad range of **well-established learning methods** and tools, for example (recurrent) neural networks, (binary, one- and true multi-class) support vector machines, algorithms for single- and multi-objective optimization, and many more.

The three design goals of Shark are **speed, modularity, and portability**. We strive for notable performance with regard to all three aspects through **object-oriented and generic programming using templated C++ code**. The library relies on only two, well-established external dependencies, namely the Boost C++ Libraries and the CMake build system. The former ensures **compatibility with the upcoming C++0x** standard; allows for **seamless integration of optimized math libraries** for again faster linear algebra computations; and also makes it easy to use **multi-core- and GPU-capable parallel processing** directives, among others.

The Shark Library is released under a **GPL license** (version 3 or later).

1) Development goals (background, aims etc.)

Machine learning is a branch of computer science and applied statistics that is concerned with automated knowledge extraction from data [Bishop, 2006; Hastie et al., 2009]. Machine learning algorithms can be characterized as software that allows systems to improve their performance on a given task based on sample data or experience. In general, machine learning algorithms have applications in countless fields such as computational biology, medical technology, computer vision, robotics, finance, search engines, spam filters, and many more.

Our design goal for the Shark library is a generic and natural framework for machine learning algorithms. The library should not "merely" offer valuable features in itself, but also substantially facilitate design and implementation of new learners (i.e., learning machines/algorithms) through its overall structure. In other words, we strive to provide a framework in which implementing a new learner is as easy as defining a single, for example, data-based loss function, which in turn can be naturally integrated with any of the existing models, optimizers, or other typical building blocks. At the same time, we aim for state-of-the-art performance in terms of computational speed. Finally, we want Shark to be portable across different platforms and compilers. Our design goals are summarized by the three keywords "speed, modularity, and portability".

The Shark code base has a history of over 15 years at the Institut für Neuroinformatik at Ruhr-University Bochum, Germany. A previous version was featured in Volume 9 of the Journal of Machine Learning Research [Igel et al., 2008]. It was the first general machine learning library accepted by this flagship journal, which is the most prestigious label open-source machine learning software can get. However, as the Shark project was started more than 15 years ago, the software architecture of Shark 2.x has not been reflecting the state-of-the-art. Therefore, the library needed to be completely rewritten.

The present submission constitutes a **complete from-scratch rewrite** of the previous Shark code base, **adding many new features and design concepts**, and vigorously unifying previously unconnected parts of the library under one common umbrella of design.

2) System configuration

Supported hardware and operating systems

The Shark library relies on CMake as a portable build system, as well as on a fully portable, standard compliant code base. This enables us to support (at least) the three major operating systems Unix, Mac OS X, and Windows together with a broad range of associated compilers. Both 32- and 64-bit platforms are supported.

Library Structure

The Shark main directory encompasses `include/` and `src/` directories for the header and source files, respectively. On the same level, we also find the folders `doc/`, `Test/` and `examples/`, which hold the Shark documentation, Shark unit test files, and Shark usage example files, respectively. The remaining folders either contain binaries, third-party material, or build information. The `include/` folder mirrors the basic library structure, which is divided into several core components (`Core`, `LinAlg`, `Data`, `Rng`, `Statistics`) as well as specific branches pertaining to different aspects in the machine learning setting (`Algorithms`, `Fuzzy`, `Models`, `ObjectiveFunctions`).

To get a first glimpse into Shark's well thought-out modular structure, let us consider the interplay between optimization and model training. In Shark 3 all optimization problems inherit the `AbstractObjectiveFunction` base class, and all optimization strategies (e.g., gradient-based or direct search methods) are derived from `AbstractOptimizer`. At this general level there is no explicit link to machine learning problems yet, such that it is easy to use the library for any type of optimization problem. Of course, the available methods are of relevance for machine learning.

Next consider the `OptimizationTrainer` class. It encapsulates an iterative model training procedure by means of three objects: (i) a `SupervisedObjectiveFunction` (typically an `ErrorFunction`, which in turn may be a data-coupled instance of an `AbstractLoss` or `AbstractCost` function, and may also possibly be combined with a regularizer); (ii) an optimizer, and (iii) a stopping criterion.

The last component is encapsulated by the `AbstractStoppingCriterion` interface. This overall setup allows the `OptimizationTrainer` to make model training based on iterative optimization a single opaque step, while at the same time relying on infrastructure for generic optimization of an objective function.

Implemented Algorithms

Among others, Shark 3 currently supports:

- Supervised learning
    - Linear discriminant analysis (LDA), Fisher–LDA
    - Linear regression
    - Support vector machines (SVMs) [Cortes and Vapnik, 1995] for one-class, binary and true multi-category classification [Dogan et al., 2011] as well as regression (the Shark SVM is the only SVM package implementing the fastest SMO-based learning algorithm for dense large-scale problems using hybrid maximum gain working set selection [Glasmachers and Igel, 2006])
    - Feed-forward and recurrent multi-layer artificial neural networks [Bishop, 1995]
    - Radial basis function networks [Bishop, 1995]
    - Regularization networks [Poggio and Smale, 2003] as well as basic Gaussian processes [Rasmussen and Williams, 2006] for regression
    - Iterative nearest neighbor classification and regression [Border, 1990]
- Unsupervised learning
    - Principal component analysis
    - Restricted Boltzmann machines [Hinton and Salakhutdinov, 2006] (including many state-of-the-art learning algorithms)
    - Hierarchical clustering
    - Data structures for efficient distance-based clustering
- Evolutionary algorithms
    - Single-objective optimization (e.g., CMA–ES [Hansen and Ostermeier, 2001; Igel et al., 2007; Suttorp et al., 2009])
    - Multi-objective optimization (in particular, highly efficient algorithms for computing as well as approximating the contributing hypervolume [Bringmann and Friedrich, 2008; Bringmann and Friedrich, 2009])
- Fuzzy systems
- Basic linear algebra and optimization algorithms (e.g., Rprop [Riedmiller, 1994; Igel and Hüsken, 2003])

Software Quality Management

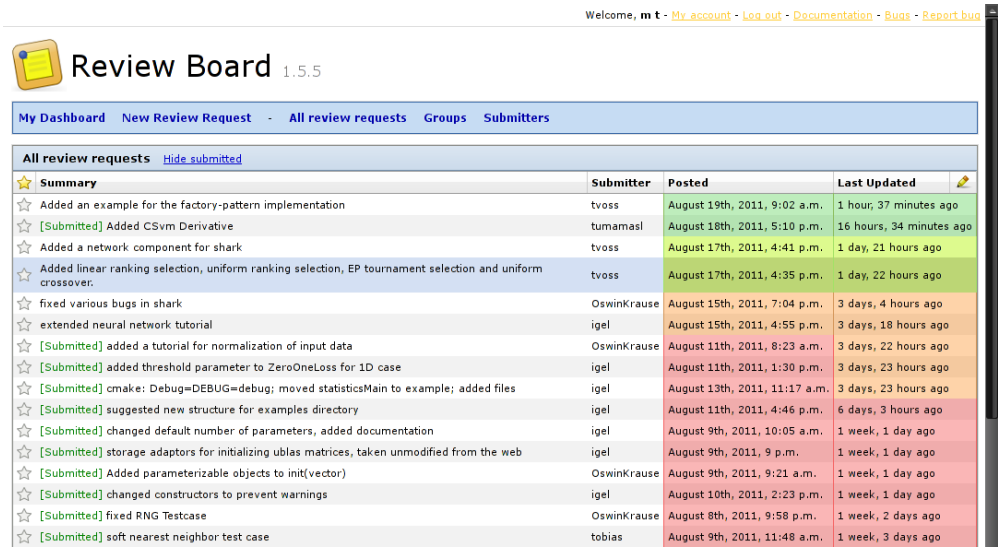


Figure 1: The pre-commit system ReviewBoard for the Shark library.

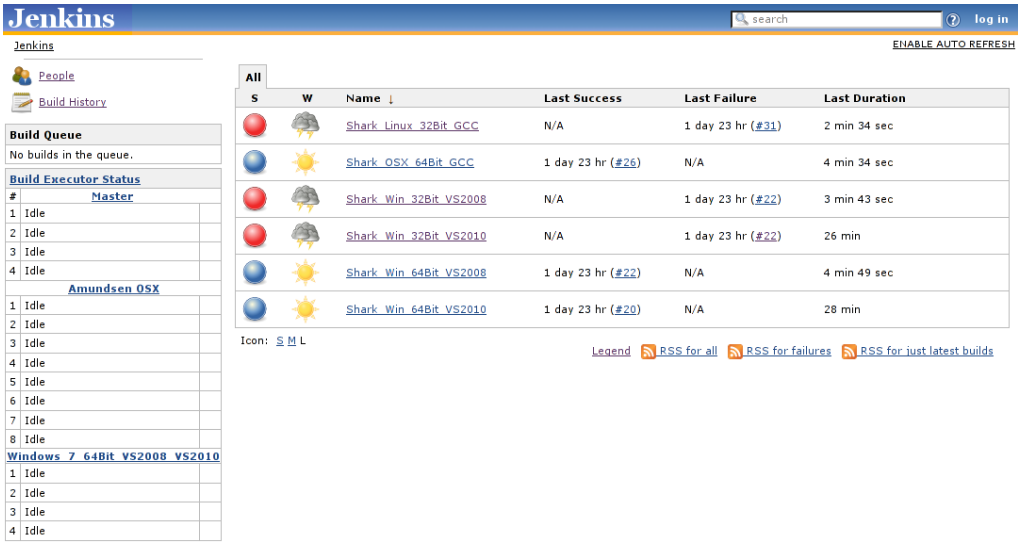


Figure 2: The continuous integration system Jenkins for the Shark library.

The Shark library deploys a range of sophisticated methods and technologies to ensure a high level of software quality and to prevent from regressions. First and foremost, an extensive and ever-growing set of unit tests is available that reflects our approach of test-driven development. Moreover, a pre-commit review system is in place (see Figure 1), and every check-in to the central subversion repository triggers a built-test-package cycle for all of our supported platforms on our

continuous integration system (see Figure 2). In addition we execute static code analysis to check for errors and to enforce our coding conventions

Deployment

Shark is available as a precompiled Debian package, as a precompiled binary package for Microsoft Windows, and as a tarball for compilation from source. The installation procedure is well covered by corresponding documentation pages.

3) Menus

Shark does not offer a graphical user interface, and there are no plans to establish one in the future. Our focus is on providing fast implementations of learning algorithms and a flexible framework for creating new ones. A typical application using Shark is a C++ program that sequentially calls different classes and functions of the Shark C++ library. While one could imagine using a graphical user interface for defining such a sequence, this in our case would simply be equivalent to a graphical tool for generating C++ code. Similarly, Shark does not offer a graphical user interface to visually explore the input data or the results generated by Shark. For this, we recommend existing open-source high-performance plotting and visualization libraries.

However, Shark will support ExtJS, a cross-browser JavaScript framework for building internet applications, which can be used for visualization independent of the operating system. Rudimentary support for ExtJS has already been added to Shark 3 and proof of concept prototypes exist. To this end, an HTTP server component will be incorporated with the Shark library that allows for making available different parts of the library (objective functions, optimizers or whole experiments) on the network via RESTful APIs.

4) Language used for development

Shark is developed purely in C++. In our view, C++ provides an optimal trade-off between higher-level functionality and speed. At the time of this writing, we do not rely on any features defined as a part of C++0x. Some components, as for example those segments of the GLKP library shipped with Shark, rely on pure C code. Similarly, the functionality offered by the Boost uBLAS library can be taken over by other implementations of the BLAS standard, which may rely on different programming languages. In order to extend the Shark user base, the construction of Python (or, more general, SWIG) bindings is one of the goals for future development.

5) Systems used

Hardware

Continuous integration server: Every commit to the Shark subversion repository triggers a build-test-package cycle of the overall library for all our supported platforms. The continuous integration system is running on a custom server machine featuring 16 compute cores powered by 2 Intel Xeon CPUs. As the overall build infrastructure is virtualized, 24GB of memory are deployed to render the system as performant as possible.

Pre-commit review server: The pre-commit review system is deployed on an off-the-shelf Dell server running Ubuntu 10.04 and an Apache/Python/MySQL setup.

Web hosting: The Shark homepage as well as the development repository are hosted by the free software portal "Sourceforge.net".

Applications: The hardware on which to run Shark for actual applications will naturally vary with the nature and scale of each respective problem. While simple algorithms on small-scale problems may finish in fractions of a second on a single core, more competitive tasks like training Boltzmann machines or high-dimensional optimization may run for weeks on a cluster of cores. For example, we successfully carried out numerous long-term experiments on the compute cluster of the “Multimodal Computing and Interaction” cluster of excellence located at the Universität des Saarlandes, Saarbrücken.

Software

Dependencies: We keep Shark as self-contained as possible. The Shark 3 package depends only on the Boost C++ libraries. For building the Shark library, CMake is used. For generating the documentation, which need not be done by end users, documentation generation tools are required (see below).

Detailed overview over third-party code and tools:

The Boost libraries: Shark relies on the Boost C++ libraries (www.boost.org) for a number of tasks such as efficient linear algebra operations on dense and sparse data structures and serialization. These well-established libraries provide stable solutions to many standard problems and are themselves extremely portable. Thus, the dependency on Boost is expected to even further increase the portability of Shark. The Boost libraries are available under the Boost software license, which is compatible with the GNU General Public License.

CMake: We use the CMake build system (www.cmake.org) to compile Shark and its accompanying example and unit test programs. CMake is quickly becoming a widespread standard and supports many platforms. CMake is available under the 3-clause (new/modified) BSD license.

GLPK: Shark for convenience ships with the simplex solver from GLPK, the GNU Linear Programming Kit (www.gnu.org/software/glpk), version 4.45. GLPK is available under the GNU General Public License.

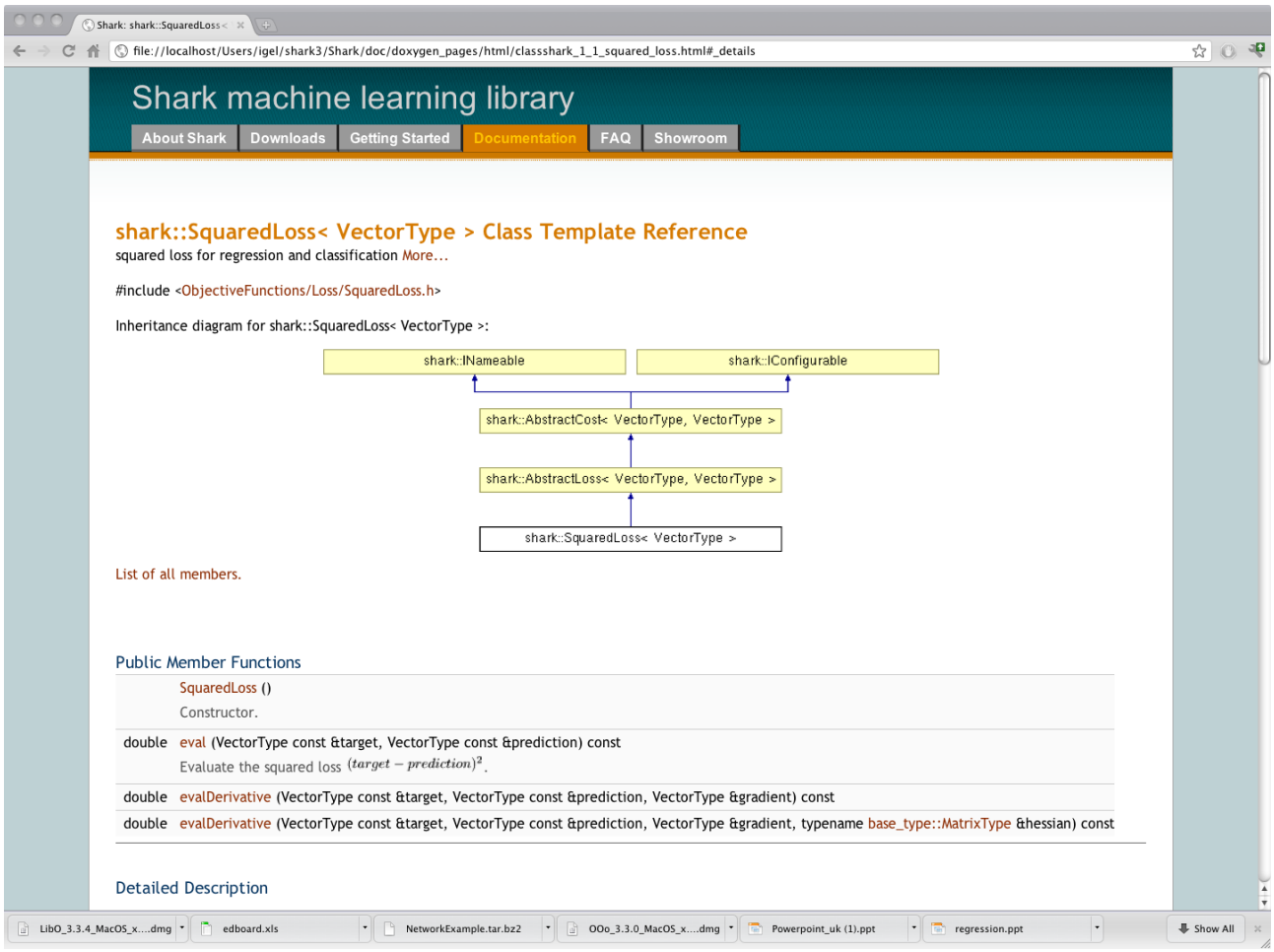


Figure 3: Class documentation using Doxygen.

Documentation: Shark mainly uses the de-facto standard Doxygen (GPL licensed, www.doxygen.org) for C++ in-code documentation extraction, see Figure 3. The Shark tutorials (see Figure 4 for an example) are converted from reStructuredText markup to html via Sphinx (BSD licensed, http://sphinx.pocoo.org), and interface the Doxygen documentation via Doxylink (BSD licensed, http://pypi.python.org/pypi/sphinxcontrib-doxylink).

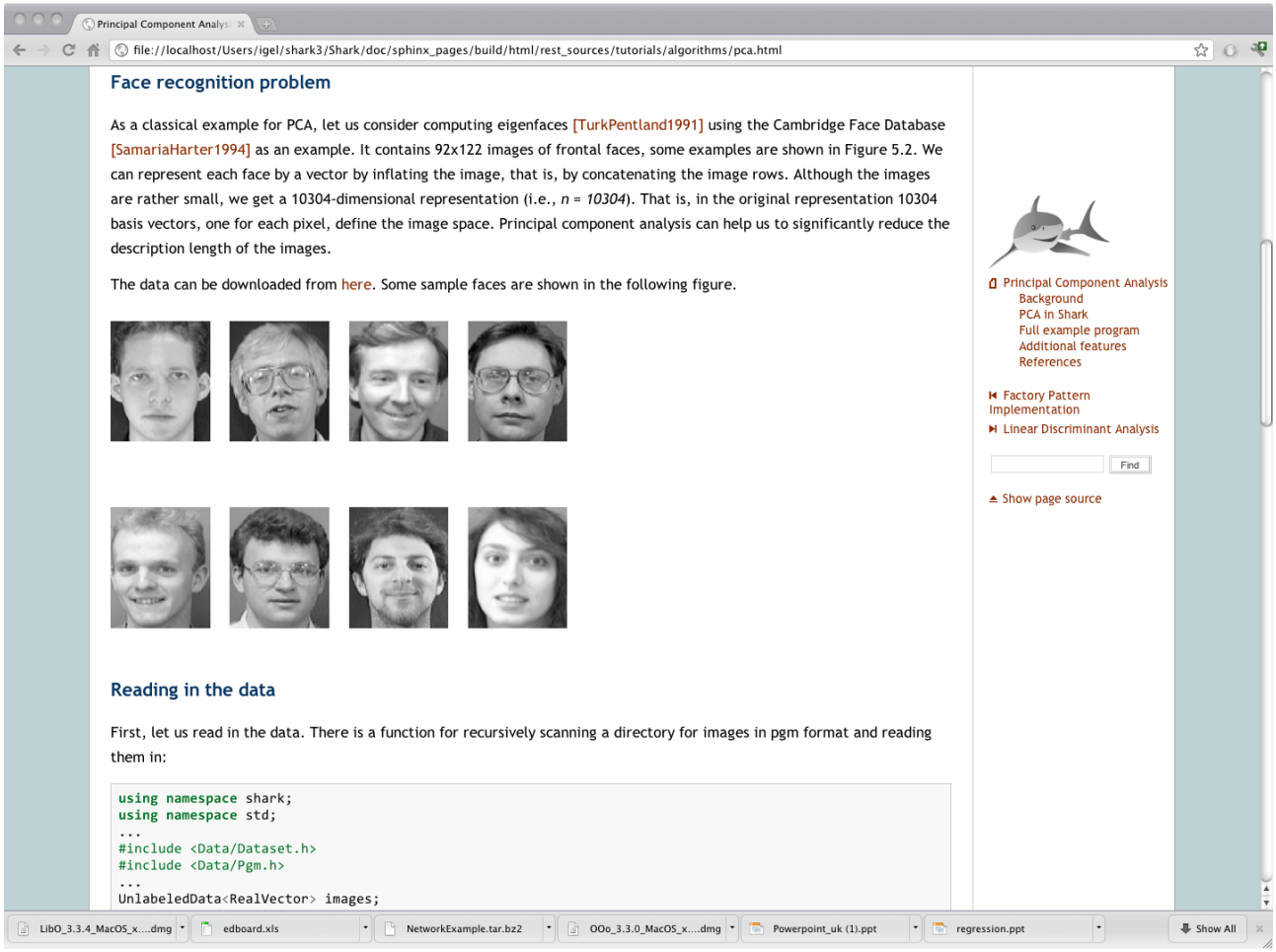


Figure 4: One of the tutorials explaining how to use the library.

Documentation & learning how to use the library

All functionality can be looked up in the documentation generated by Doxygen. An example from the class list is shown in Figure 3. There are several ways to learn how to use the Shark library. It comes with online tutorials that explain the principles of the library and show how to apply it in practice. An example is shown in Figure 4. Further, the software is shipped with example

programs, which can be found in the subdirectory `examples/`. The unit tests in the subdirectory `Test/` serve as additional examples of how to use the various components of the library.

6) Plan for each development stage

This present version of the Shark library has been completely re-written from the previous code base over the last 10 months. The structure of the library has been changed drastically. All the measures for software quality measurement outlined above have newly been added. To the Open Source Software World Challenge 2011, we submit an alpha pre-release version of the library. The goals for the remaining months of 2011 are:

- A better integration of the component for Fuzzy Systems.
- Improving the documentation, in particular adding more tutorials and example programs.
- Implementing some remaining features of the previous release that are not yet ported, especially the approximation of kernel expansions [Romdhani, 2004; Suttorp and Igel, 2007], k-means clustering, and missing components of the evolutionary algorithms toolbox.
- Implementing maximum likelihood model selection for binary SVMs [Glasmachers and Igel, 2010].
- Optimization of the kernel-target alignment for (SVM) model selection [Igel et al., 2007].
- Extending the RBM (Restricted Boltzmann Machine) component.
- Not all parts of the library yet fully adhere to the new Shark coding and documentation standards. These will be adapted accordingly.

After these steps, the full release of the all-new Shark 3.0 will be carried out. For the long-term development stages see part 2. of this proposal.

7) Number of personnel input and work assignment

Please refer to the enclosed "List of Team Members" for the names and respective main work areas of all current Shark developers. The core team consists of six developers, for which developing the Shark library mostly coincides with their full-time academic work on machine learning algorithms. In addition, a number of former Shark developers as well as regular users are contributing on a smaller scale. The team members' specialized work areas follow the division of the library into topical categories, cf. item 2).

2. Long-term prospects of the program developed (No specific form is required.)

Even before the current from-scratch rewrite, the Shark library (or its predecessor code base), had an established history of long-term continuation and support, dating back over 15 years at the Institut für Neuroinformatik at the Ruhr-University in Bochum, Germany. The Shark library's popularity has constantly risen over the past, with over 500 downloads in March 2011. For the future, we plan to:

- Increase the interaction with the user community and the number of contributing external developers.
- Add new learning algorithms to the library, in particular decision trees, random forests, as well as boosting and transfer learning algorithms.
- Complete the transfer of our reinforcement learning code base to Shark 3.
- Extend the Shark user base by providing bindings for other programming languages, preferably via the SWIG interface generator tools.
- Further simplify the usability of generic multi-core and GPU processing directives for linear algebra components in Shark. Moreover, stochastic simulations of approximation algorithms, e.g., approximation of the least hypervolume contributor, on behalf of special-purpose processors are part of our mid-term and long-term plans for the library.
- Offer Shark tutorials at machine learning conferences and summer schools.

References:

Bishop, C. M. (2006), *Pattern Recognition and Machine Learning*, Springer-Verlag.

Bishop, C. M. (1995), *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford.

Border, A. J. (1990), 'Strategies for efficient incremental nearest neighbor search', *Pattern Recognition* **23**(1/2), 171--178.

Bringmann, K. & Friedrich, T. (2009), Approximating the Least Hypervolume Contributor: NP-Hard in General, But Fast in Practice, in '*Proc. of the 5th International Conference on Evolutionary Multi-Criterion Optimization* (EMO 2009)', Springer-Verlag, , pp. 6-20.

Bringmann, K. & Friedrich, T. (2008), Approximating the Volume of Unions and Intersections of High-Dimensional Geometric Objects, in '*Proc. of the 19th International Symposium on Algorithms and Computation* (ISAAC)', Springer-Verlag, , pp. 436-447.

Cortes, C. & Vapnik, V. (1995), 'Support-Vector Networks', *Machine Learning* **20**(3), 273-297.

Dogan, Ü.; Glasmachers, T. & Igel, C. (2011), 'Fast Training of Multi-Class Support Vector Machines', Technical report, Department of Computer Science, University of Copenhagen.

Glasmachers, T. & Igel, C. (2010), 'Maximum Likelihood Model Selection for 1-Norm Soft Margin SVMs with Multiple Parameters', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **32**(8), 1522-1528.

Glasmachers, T. & Igel, C. (2006), 'Maximum-Gain Working Set Selection for Support Vector Machines', *Journal of Machine Learning Research* **7**, 1437-1466.

Hansen, N. & Ostermeier, A. (2001), 'Completely Derandomized Self-Adaptation in Evolution Strategies', *Evolutionary Computation* **9**(2), 159-195.

Hastie, T.; Tibshirani, R. & Friedman, J. (2009), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer-Verlag.

Hinton, G. E. & Salakhutdinov, R. R. (2006), 'Reducing the Dimensionality of Data with Neural Networks', *Science* **313**(5786), 504--507.

Igel, C.; Glasmachers, T. & Heidrich-Meisner, V. (2008), 'Shark', *Journal of Machine Learning Research* **9**, 993-996.

Igel, C.; Glasmachers, T.; Mersch, B.; Pfeifer, N. & Meinicke, P. (2007), 'Gradient-based Optimization of Kernel-Target Alignment for Sequence Kernels Applied to Bacterial Gene Start Detection', *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **4**(2), 216-226.

Igel, C. & Hüsken, M. (2003), 'Empirical Evaluation of the Improved Rprop Learning Algorithm', *Neurocomputing* **50**(C), 105-123.

Igel, C.; Hansen, N. & Roth, S. (2007), 'Covariance Matrix Adaptation for Multi-objective Optimization', *Evolutionary Computation* **15**(1), 1-28.

Poggio, T. & Smale, S. (2003), 'The mathematics of learning: Dealing with data', *Notices of the AMS* **50**(5), 537--544.

Rasmussen, C. E. & Williams, C. K. I. (2006), *Gaussian Processes for Machine Learning*, MIT Press.

Riedmiller, M. (1994), 'Advanced Supervised learning in Multi-layer Perceptrons – From Backpropagation to Adaptive Learning Algorithms', *Computer Standards and Interfaces* **16**(5), 265-278.

Romdhani, S.; Torr, P.; Schölkopf, B. & Blake, A. (2004), 'Efficient face detection by a cascaded support-vector machine expansion', *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **460**(2051), 3283-3297.

Suttorp, T.; Hansen, N. & Igel, C. (2009), 'Efficient Covariance Matrix Update for Variable Metric Evolution Strategies', *Machine Learning* **75**(2), 167-197.

Suttorp, T. & Igel, C. (2007), Resilient Simplification of Kernel Classifiers, in J. Marques de Sá; L. A. Alexandre; W. Duch & D. P. Mandic, ed., '*International Conference on Artificial Neural Networks (ICANN 2007)*', Springer-Verlag, pp. 139-148.