TunedIT.org: System for Automated Evaluation of Algorithms in Repeatable Experiments

Marcin Wojnarski[1,2], Sebastian Stawicki[2], and Piotr Wojnarowski[1,2]

[1] TunedIT Solutions
Zwirki i Wigury 93 lok. 3049, 02-089 Warszawa, Poland
[2] Faculty of Mathematics, Informatics and Mechanics, University of Warsaw
Banacha 2, 02-097 Warszawa, Poland

Abstract. In this paper we present TunedIT system which facilitates evaluation and comparison of machine-learning algorithms. TunedIT is composed of three complementary and interconnected components: TunedTester, Repository and Knowledge Base.

TunedTester is a stand-alone Java application that runs automated tests (experiments) of algorithms. Repository is a database of algorithms, datasets and evaluation procedures used by TunedTester for setting up a test. Knowledge Base is a database of test results. Repository and Knowledge Base are accessible through TunedIT website. TunedIT is open and free for use by any researcher. Every registered user can upload new resources to Repository, run experiments with TunedTester, send results to Knowledge Base and browse all collected results, generated either by himself or by others.

As a special functionality, built upon the framework of automated tests, TunedIT provides a platform for organization of on-line interactive competitions for machine-learning problems. This functionality may be used, for instance, by teachers to launch contests for their students instead of traditional assignment tasks; or by organizers of machine-learning and data-mining conferences to launch competitions for the scientific community, in association with the conference.

1 Introduction

Almost every paper published in the field of machine learning contains experimental section, which presents empirical analysis, evaluation and comparison of described algorithms. We investigated 81 regular research papers published in Volume 9/2008 of Journal of Machine Learning Research, excluding articles assigned to special topics or to the Machine Learning Open Source Software track, as well as responses to other papers. We observed that as much as 75 (93%) of the papers contained experimental section. Experimental results constitute ultimate proof of strengths of described methods, so even a slight improvement in the methodology of conducting experiments would be beneficial to the whole community of machine learning researchers and facilitate design of even better algorithms.

Currently used experimental methodology has serious weaknesses. The biggest one is that experiments performed by the author of a new algorithm and described in a research article cannot be reproduced by other researchers. On paper, the author should provide full details of the algorithm and experimental procedure, sufficient to repeat the experiment. In practice:

– Providing all the details would make the article unreadable and substantially longer, so the description is rarely complete. For example, if experiments involve decision models with adaptively optimized parameters, like weights of neural networks, it is rarely described in detail how the parameters are initialized at the beginning of the training process.
– The author himself may be unaware of some important details, for example implementation bugs.
– Reimplementation of the algorithm by another researcher could take weeks of work and thus is not feasible, even if theoretically possible.
– Even if the author made implementation of the algorithm publicly available, significant effort must be put into learning how to use the implementation.
– Recreation of experimental setup may be difficult, even when all necessary elements of the experiment – data sets, implementations of algorithms etc. – are available.
– There is high risk of human mistakes. They are very hard to detect, because usually the outcomes of experiments are just numbers, only slightly different between each other. Many types of mistakes are very hard to notice.

To address these problems we designed and implemented TunedIT system (http://tunedit.org) – an integrated platform for automated evaluation of machine learning algorithms. Thanks to automation, TunedIT enables researchers to design and execute experiments that are fully repeatable and generate reproducible results. This system is presented in the following sections.

Previous attempts to address the aforementioned problems include: Delve software environment for evaluation of learning algorithms in valid experiments [4,3]; Experiment Databases for Machine Learning[1] (ExpDB) for collecting and sharing experimental results [1]; Machine Learning Open Source Software[2] (MLOSS) website for sharing implementations [5]; UCI Machine Learning Repository[3] for collecting and sharing datasets [2]; Computational Intelligence and Machine Learning (CIML) Community Portal[4] [7].

2 TunedIT System

TunedIT system combines three interrelated components (Fig. 1):

1. *TunedTester*: a Java application for automated evaluation of algorithms according to test specification provided by the user.

[1] http://expdb.cs.kuleuven.be/
[2] http://mloss.org/
[3] http://www.ics.uci.edu/∼mlearn/
[4] http://www.cimlcommunity.org/

2. *Repository*: a database of machine learning resources. These include algorithms, datasets and evaluation procedures, which can be used by TunedTester to set up and execute experiments.
3. *Knowledge Base* (KB): a database of test results. On user's request, TunedTester may send results of tests to TunedIT. Here, results submitted by different researchers are merged into rich and comprehensive Knowledge Base that can be easily browsed for accurate and thorough information on specific algorithms or datasets.

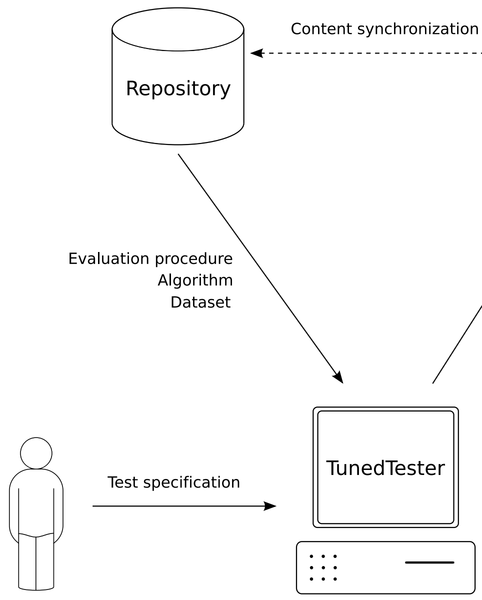

Fig. 1. Main components of TunedIT system and their interactions. *Repository*: a database of machine learning resources. *Knowledge Base*: a database of test results. *TunedTester*: an automated testing application. TunedTester takes a test specification from the user, downloads resources from Repository needed to set up the test, executes the test and sends the result together with the test specification to Knowledge Base. Additionally, upon each modification of the contents of Repository, such as deletion of a resource, the contents of Knowledge Base is synchronized accordingly, so that the results collected in Knowledge Base are always consistent with the current contents of Repository.

Repository and Knowledge Base reside on a server and can be accessed through TunedIT website at the following URL: http://tunedit.org. All registered users can upload resources to Repository, browse results collected in Knowledge Base and submit new results generated by TunedTester. Registration is open to everyone. TunedTester runs on a client computer, which typically is the user's local machine, and communicates with Repository and Knowledge Base through Internet.

2.1 Repository

Repository is a database of files – *resources* – related to machine learning and data mining. In particular, these include datasets, code of algorithms and evaluation procedures. Repository is located on TunedIT server and is accessible for all registered users – they can view and download resources, as well as upload new ones. The role of Repository in TunedIT is three-fold:

- It serves as a collection of algorithms, datasets and evaluation procedures that can be downloaded by TunedTester and used in tests.
- It provides space where users can share ML and DM resources with each other.
- It constitutes a context and point of reference for interpretation of results generated by TunedTester and logged in Knowledge Base. For instance, when the user is browsing KB and viewing results for a given test specification, he can easily navigate to corresponding resources in Repository and check their contents, so as to validate research hypotheses or come up with new ones. Thus, Repository is not only a convenient tool that facilitates execution of tests and sharing of resources, but - most of all - secures interpretability of results collected in Knowledge Base.

Repository has similar structure as a local file system. It contains a hierarchy of folders, which in turn contain files - resources. Upon registration, every user is assigned *home folder* in Repository's root folder, with its name being the same as the user's login. The user has full access to his home folder, where he can upload/delete files, create subfolders and manage access rights for resources. All resources uploaded by users have unique names (access paths in Repository) and can be used in TunedTester exactly in the same way as preexisting resources.

Access rights. Every file or folder in Repository is either *public* or *private*. All users can view and download public resources. Private files are visible only to the owner, while to other users they appear like if they did not exist - they cannot be viewed nor downloaded and their results do not show up at KB page. Private folders cannot be viewed by other users, although subfolders and files contained in them can be viewed by others, given that they are public themselves. In other words, the property of being private does not propagate from a folder to files and subfolders contained inside.

2.2 TunedTester

TunedTester (TT) is a Java application that enables fully automated evaluation of algorithms, according to test specification provided by the user. Single run of evaluation is called a *test* or *experiment* and corresponds to a triple of resources from Repository:

1. **Algorithm** is the subject of evaluation.
2. **Dataset** represents an instance of a data mining problem to be solved by the algorithm.
3. **Evaluation procedure** (EP) is a Java class that implements all steps of the experiment and, at the end, calculates a quality measure.

It is worth to note that the evaluation procedure is not hard-wired into TunedTester but is a part of test configuration just like the algorithm and dataset. Every user can implement new evaluation procedures to handle new kinds of algorithms, data types, quality measures or data mining tasks. In this way, TunedTester provides not only full automation of experiments, but also high level of flexibility and extendability.

TunedTester runs locally on user's computer. All resources that comprise the test are automatically downloaded from Repository. If requested, TunedTester can submit results of tests to Knowledge Base. They can be analysed later on with convenient web interface of KB.

All TunedIT resources are either files, like `UCI/iris.arff`, or Java classes contained in JAR files, like

`Weka/weka-3.6.1.jar:weka.classifiers.lazy.IB1` .

Typically, datasets have a form of files, while evaluation procedures and algorithms have a form of Java classes. For datasets and algorithms this is not a strict rule, though. To be executable by TunedTester, evaluation procedure must be a subclass of

`org.tunedit.core.EvaluationProcedure`

located in `TunedIT/core.jar` file in Repository. `TunedIT/core.jar` contains also

`ResourceLoader` and `StandardLoader`

classes, which can be used by the evaluation procedure to communicate with TunedTester environment and read the algorithm and dataset files. It is up to the evaluation procedure how the contents of these files is interpreted: as bytecode of Java classes, as a text file, as an ARFF, CSV or ZIP file etc. Thus, different evaluation procedures may expect different file formats and not every evaluation procedure must be compatible with a given algorithm or dataset. This is natural, because usually the incompatibility of file formats is just a reflection of more inherent incompatibility of resource types. There are many different types of algorithms – for classification, regression, feature selection, clustering etc. – and datasets – time series, images, graphs etc. – and each of them must be evaluated differently anyway. Nonetheless, it is also possible that the evaluation procedure supports several different formats at the same time.

Dataset file formats and algorithm APIs that are most commonly used in TunedIT and are supported by standard evaluation procedures include:

– ARFF file format for data representation. This format was introduced by Weka and became one of the most popular in machine learning community.

- Debellor's API defined by `org.debellor.core.Cell` class for implementation of algorithms.
- Weka's API defined by `weka.classifiers.Classifier` class for implementation of algorithms.
- Rseslib's API defined by `rseslib.processing.classification.Classifier` interface for implementation of algorithms.

It is also possible for a dataset to be represented by a Java class, the class exposing methods that return data samples when requested. This is a way to overcome the problem of custom file formats. If a given dataset is stored in atypical file format, one can put it into a JAR file as a Java resource and prepare a wrapper class that reads the data and returns samples in common representation, for example as instances of Debellor's `Sample` class.

Users may implement evaluation procedures that support any other file formats or algorithm APIs, not mentioned above. We also plan to extend this list in the future, so that basic evaluation procedures created by us can handle other formats and APIs.

More importantly, we would like to extend TunedTester with support for other programming languages, not only Java. Although this task will be more laborious, it is important for all the researchers who do not use Java for implementation of their algorithms.

Test specification. Test specification is a formal description for TunedTester of how the test should be set up. It is a combination of three identifiers – TunedIT *resource names* – of TunedIT resources which represent an evaluation procedure, an algorithm and a dataset that will be employed in the test:

$$\textit{Test specification} = \textit{Evaluation procedure} + \textit{Algorithm} + \textit{Dataset}$$

TunedIT resource name is a full access path to the resource in Repository, as it appears on Repository page. It does not include leading slash "/". For example, the file containing Iris data and located in UCI folder[5] has the following name:

`UCI/iris.arff`

Java classes contained in JARs are also treated as resources. TunedIT resource name of a Java class is composed of the containing JAR's name followed by a colon ":" and full (with package) name of the class. For instance, ClassificationTT70 class contained in `TunedIT/base/ClassificationTT70.jar` and `org.tunedit.base` package has the following name:

$$TunedIT/base/ClassificationTT70.jar : org.tunedit.base.ClassificationTT70$$

Resource names are case-sensitive.

Many algorithms expose a number of parameters that can be set by the user to control and modify algorithm's behavior. Currently, test specification does not

[5] See http://tunedit.org/repo/UCI/iris.arff

include values of parameters, and thus it is expected that the algorithm will apply default values. If the user wants to test an algorithm with non-default parameters he should write a wrapper class which internally invokes the algorithm with parameters set to some non-default values. The values must be hard-wired in the wrapper class, so that the wrapper itself does not expose any parameters. In the future we plan to add the ability to provide non-default parameter values directly in test specification.

Although the test specification has simple structure, it can represent a broad range of different tests. This is because the contents of the algorithm and dataset is interpreted by evaluation procedure, which is pluggable itself. If some kind of algorithm or dataset is not handled by existing evaluation procedures, a new procedure can be implemented for them. Thus, pluggability of evaluation procedures gives the power to test any kinds of algorithms on any kinds of data, while keeping the results interpretable thanks to simple and consistent test specifications.

Sandbox. Users of TunedTester may safely execute tests of any algorithms present in Repository, even if the code cannot be fully trusted. TunedTester exploits advanced features of Java Security Architecture to assure that the code executed during tests do not perform any harmful operation, like deleting files on disk or connecting through the network. Code downloaded from Repository executes in a *sandbox* which blocks the code's ability to interact with system environment. This is achieved through the use of a dedicated Java class loader and custom security policies. Similar mechanisms are used in web browsers to protect the system from potentially malicious applets found on websites.

Local cache. Communication between TunedTester and TunedIT server is efficient thanks to the cache directory which keeps local copies of resources from Repository. When the resource is needed for the first time and must be downloaded from the server, its copy is saved in the cache. In subsequent tests, when the resource is needed again, the copy is used instead. In this way, resources are downloaded from Repository only once. TunedTester detects if the resource has been updated in Repository and downloads the newest version in such case. Also, any changes introduced to the local copies of resources are detected, so it is not possible to run a test with corrupted or intentionally faked resources.

2.3 Knowledge Base

Knowledge Base (KB) is an open-access database containing test results from TunedTester, built collaboratively by TunedIT users. It is located on TunedIT server. Every user registered in TunedIT may submit results to KB by checking the "Send results to Knowledge Base" option in TunedTester GUI. Thanks to standardization and automation of tests, results submitted by different users are all comparable and thus can be merged together in a single database. In this way, TunedIT gives researchers an opportunity to build collectively an experiment database of unprecedented size and scope. This is impossible to achieve using previous approaches. For example, in the system by [1] only the administrators have full access to the database and can insert new results. As of January 2010, KB contains over 150,000 atomic results and 6,700 aggregated results (see below for the definition of atomic and aggregated results).

To guarantee that results in KB are always consistent with the contents of Repository and that Repository can serve indeed as a context for interpreration of the results, when a new version of resource is uploaded, KB gets automatically cleaned out of all out-dated results related to the old version of the resource. Thus, there is no way to insert results into KB that are inconsistent with the contents of Repository.

Knowledge Base has a web interface that allows for easy querying and browsing the database.

2.4 Nondeterminism of Test Results

It is very common for machine learning implementations to include nondeterministic factors. For example:

– Evaluation procedures may split data randomly into training and test parts, which yields different splits in every trial. This is the case with standard procedures available in TunedIT: ClassificationTT70 and RegressionTT70[6].
– Algorithms for training of neural networks may perform random initialization of weights at the beginning of learning.
– Data samples may be generated randomly from a given probabilistic distribution, resulting in a different dataset in each repetition of the experiment.

Consequently, experiments executed with TunedTester may also include nondeterministic factors and generate different outcomes on every run. For this reason, instead of analysing single test outcomes, we need to analyse probabilistic distribution of results produced for a given test specification. To this end, TunedIT introduces the notions of *atomic result* and *aggregated result*.

Definition 1 (Atomic result). *Atomic result is the result of a single test executed by TunedTester for a given test specification.*

Atomic result gives a snapshot of algorithm's behavior in a single random scenario of the experiment. It is possible to execute many tests for the same specification and log all their results in KB. Thus, there can be many atomic results present in KB which correspond to the same specification.

Definition 2 (Aggregated result). *Aggregated result is the aggregation of all atomic results present in KB and corresponding to a given test specification. Here, aggregation means a set of statistics: arithmetic mean, standard deviation etc.*

There can be only one aggregated result for a given specification. Aggregated results are dynamically calculated at TunedIT server and presented on KB web

[6] See http://tunedit.org/repo/TunedIT/base

page[7]. Currently, users of TunedIT do not have direct access to atomic results stored in KB, but we plan to introduce this functionality in the future. At the moment, users can only see atomic results of their own experiments, in TunedTester, printed onto output console.

Presence of nondeterminism in experiments is highly desirable. If tests are fully deterministic, they always produce the same outcome and thus the aggregated result (mean) is the same as all atomic results, with standard deviation equal to zero. With nondeterminism, experimental results give broader knowledge about the tested algorithm – non-zero deviation measures how reliably and repeatably the algorithm behaves – and more reliable estimation of its expected quality (mean of multiple atomic results which are different between each other). Therefore, when implementing new algorithms, evaluation procedures or data generators, it is worth to introduce nondeterminism, if only this does not disturb essential functionality of the implementation. For instance, if an evaluation procedure splits the data into training and test subsets, it is better to perform this split at random instead of picking every time the same samples, e.g., with the *first* 70% of samples always falling into training subset.

2.5 Security Issues: Validity of Results

The user may assume that results generated by others and collected in KB are valid, in a sense that if the user runs the same tests by himself he would obtain the same expected results. In other words, results in KB can be trusted even if their authors – unknown users of TunedIT – cannot be trusted. This is possible thanks to numerous security measures built into Repository, TunedTester and KB, which ensure that KB contents cannot be polluted neither by accidental mistakes nor intentional fakery of any user. It is also worth to note that all results from KB can be easily verified, because corresponding test specifications are known and all involved resources are present in Repository, so reproducing an experiment is a matter of launching TunedTester, typing the test specification and pressing “Run...”.

3 Conclusions

In this paper, we presented TunedIT system, which enables automated evaluation of machine-learning and data-mining algorithms; execution of repeatable experiments; sharing of experimental results and resources – datasets, algorithms, evaluation procedures – among researchers. The idea and design of such an integrated system, which supports all important aspects of experimentations in computational intelligence, is unique. TunedIT has already attracted researchers' attention and after just 5 months of functioning has 400 registered users. We hope that it will enjoy wide adoption in the scientific community and facilitate communication between researchers, exchange of ideas and design of yet better algorithms.

[7] See http://tunedit.org/results

Acknowledgements

The research has been partially supported by grants N N516 077837 and N N516 368334 from Ministry of Science and Higher Education of the Republic of Poland.

References

1. Blockeel, H., Vanschoren, J.: Experiment databases: Towards an improved experimental methodology in machine learning. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) PKDD 2007. LNCS (LNAI), vol. 4702, pp. 6–17. Springer, Heidelberg (2007)
2. Newman, D.J., Hettich, S., Blake, C., Merz, C.: UCI repository of machine learning databases (1998), `http://www.ics.uci.edu/~mlearn/MLRepository.html`
3. Neal, R.: Assessing relevance determination methods using delve. In: Neural Networks and Machine Learning, pp. 97–129. Springer, Heidelberg (1998), `http://www.cs.utoronto.ca/~radford/ftp/ard-delve.pdf`
4. Rasmussen, C.E., Neal, R.M., Hinton, G.E., van Camp, D., Revow, M., Ghahramani, Z., Kustra, R., Tibshirani, R.J.: The DELVE Manual. University of Toronto, 1.1 edn. (December 1996), `http://www.cs.utoronto.ca/~delve`
5. Sonnenburg, S., Braun, M.L., Ong, C.S., Bengio, S., Bottou, L., Holmes, G., LeCun, Y., Müller, K.R., Pereira, F., Rasmussen, C.E., Rätsch, G., Schölkopf, B., Smola, A., Vincent, P., Weston, J., Williamson, R.C.: The need for open source software in machine learning. Journal of Machine Learning Research 8, 2443–2466 (2007)
6. Wojnarski, M.: Debellor: a data mining platform with stream architecture. In: Peters, J.F., Skowron, A., Rybiński, H. (eds.) Transactions on Rough Sets IX. LNCS, vol. 5390, pp. 405–427. Springer, Heidelberg (2008)
7. Zurada, J.M., Wojtusiak, J., Chowdhury, F., Gentle, J.E., Jeannot, C.J., Mazurowski, M.A.: Computational intelligence virtual community: Framework and implementation issues. In: IJCNN, pp. 3153–3157. IEEE, Los Alamitos (2008), `http://www.mli.gmu.edu/jwojt/papers/08-5.pdf`