Improving reproducibility of data science experiments

Tatiana Likhomanenko[a,b,c] ANTARES@YANDEX-TEAM.RU
Alexey Rogozhnikov[a,b] AXELR@YANDEX-TEAM.RU
Alexander Baranov[a] SASHAB1@YANDEX-TEAM.RU
Egor Khairullin[a] MIKARI@YANDEX-TEAM.RU
Andrey Ustyuzhanin[a,b,c] ANADERI@YANDEX-TEAM.RU
[a] *Yandex Data Factory, Yandex School of Data Analysis, Moscow, Russia*
[b] *Higher School of Economics National Research University Moscow, Russia*
[c] *NRC "Kurchatov Institute", Moscow, Russia*

Abstract

Data analysis in fundamental sciences nowadays is an essential process that pushes frontiers of our knowledge and leads to new discoveries. At the same time we can see that complexity of those analyses increases fast due to a) enormous volumes of datasets being analyzed, b) variety of techniques and algorithms one have to check inside a single analysis, c) distributed nature of research teams that requires special communication media for knowledge and information exchange between individual researchers. There is a lot of resemblance between techniques and problems arising in the areas of industrial information retrieval and particle physics. To address those problems we propose Reproducible Experiment Platform (REP), a software infrastructure to support collaborative ecosystem for computational science. It is a Python based solution for research teams that allows running computational experiments on shared datasets, obtaining repeatable results, and consistent comparisons of the obtained results. Several analysis using Key features of REP are illustrated on several practical cases that were performed at LHCb experiment at CERN.

Keywords: machine learning, reproducibility, computation infrastructure, analysis preservation

1. Introduction

This paper presents Reproducible Experiment Platform (REP), which is created to perform a reproducible data analysis in a comfortable way. As Karl Popper said: "Non-reproducible single occurrences are of no significance to science".

Being a significant part of a research, scientific analyses should be prepared in a reproducible way. However, there are several reasons why many analyses prepared in recent years have difficulties in reevaluation: keeping data, code and results together; independence on development platform; availability of different algorithms; distributed research; repeatable data preprocessing. The REP toolkit addresses these problems (will be discussed further) and, moreover, enables researches to collaborate by sharing analysis code and results over the Internet. This platform provides methods for preparing and processing data, ways to use different machine learning techniques. Yet, we are hoping to have readable and, which is important, checkable code using the platform.

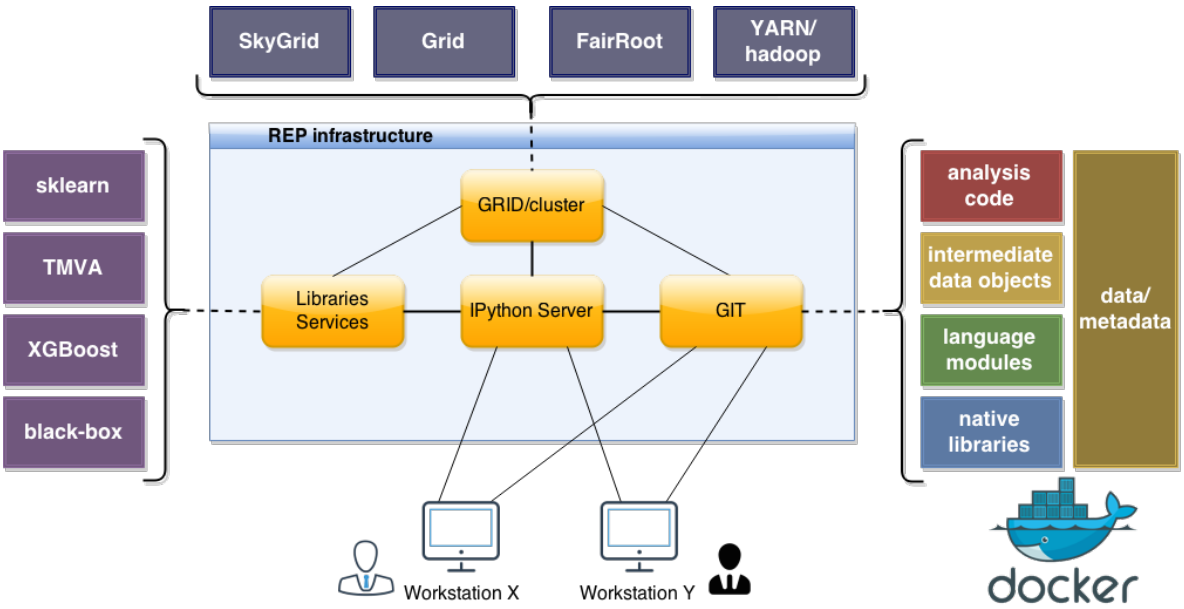

Figure 1: REP infrastructure. IPython notebook is an analysis environment, in which users work together. In the notebook we use different machine learning libraries (left) for data analysis, parallel systems to speedup optimal model search (top), `git` to store all code, data and results (right).

2. REP Infrastructure

What are the main points we expect from an analysis platform? Firstly, we need some interactive environment for fast experiments with data. Secondly, resulting code should be simple and reproducible. Thirdly, it is necessary to keep data, code and results together. REP covers all three features.

Interactive Python Fernando Perez et al. (2007), or IPython notebook, is used as an analysis environment (see Figure 1) in REP. For intellectual data analysis we use different machine learning libraries, which are popular among data scientists. For this purpose REP contains wrappers over algorithms from different machine learning libraries, which provide `scikit-learn` interface F. Pedregosa et al. (2011). A typical analysis task, search for optimal predictive model, can be speeded up using parallel computational system. In Figure 1 they are presented as a GRID/cluster system. One of the parallel execution systems is provided out-of-the-box by IPython (IPython cluster). Another significant part of experiments is `git` S. Chacon, B. Straub (2014), version control system, which stores all code, corresponding results, trained models and data.

One of the most significant problems for reproducibility is keeping track of versions of all libraries used (and their numerous dependencies). This can be archived by using virtual machine, where a scientist saves his analysis with all dependencies. However, a better option exists today: one can use like a light-weight virtual machine, Docker container `https://www.docker.com`. The images of virtual machines can be combined together inside a

Docker to provide possibilities given by different containers. It has several other advantages, among which incremental versioning of containers. This versioning implies that to change version of container user doesn't need to reload complete image, only some 'update' part is downloaded. That is why we provide a Docker container with REP and all it dependencies. Being a virtual machine, the REP container is expected to work after many years in exactly the same way as it worked at the time of creation on variety of operating systems supporting virtualization.

3. Machine Learning Pipelines

The main feature of REP is support for different machine learning libraries. Wrappers over all libraries use scikit-learn classifier interface because of its popularity in data science community and its convenience. Moreover, there are other advantages: support of single interface makes it possible to use ensembling algorithms. For example, REP provides TMVA A. Hoecker et al. (2007) wrapper in scikit-learn interface and ensembling algorithm over any TMVA method can be constructed. One can construct scikit-learn AdaBoost over TMVA rectangular cut method, or TMVA multilayer perceptron, or any another TMVA method. This way of combining different methods is very typical for scikit-learn.

Wrappers over machine learning libraries are basic elements to construct complicated analysis scheme using ensembling algorithms or another hierarchical training models. Basic REP building blocks are:

- estimators — wrapper over algorithms from variety of libraries matching fit/predict interface, that build classification or regression models http://yandex.github.io/rep/estimators.html
- metrics (including user-defined)
- factory for training and comparing several estimators (see http://yandex.github.io/rep/metaml.html#module-rep.metaml.factory)
- grid search, supporting various optimization algorithms (at the moment there are several algorithms: random sampling, Metropolis-like optimization, regression-based optimization, annealing optimization, see http://yandex.github.io/rep/metaml.html#module-rep.metaml.gridsearch).

At the moment there are wrappers over such libraries as: scikit-learn, XGBoost https://github.com/dmlc/xgboost, TMVA, theanets https://github.com/lmjohns3/theanets, pybrain https://github.com/pybrain/pybrain, neurolab https://github.com/zueve/neurolab. REP can incorporate service-based classifiers (e.g. Event Filter that is a web-service for machine learning provided by Yandex, which is available only for CERN researchers. Event Filter uses MatrixNet algorithm A. Gulin et al. (2011) developed at Yandex).

By means of REP it is easy to construct ensembling and hierarchical models using basic elements: scikit-learn AdaBoost ensemble algorithm, bagging, folding, *etc.* Any wrapper can be base estimator for any of these hierarchical models. It is necessary to find the best model among of all these configurations. For this purpose REP provides a set of grid

search-like algorithms, which takes any estimator and looks for the best estimator's hyper parameters by optimizing quality metric selected. The grid search can optimize parameters for all hierarchical levels of complicated estimator (for instance, this allows tuning number of stages in AdaBoost and parameters of base classifier used by AdaBoost). One more task frequently arising is different models training and comparison of their performance. Specially for these purposes REP contains a factory. To speed up training operations parallel system (IPython cluster, threads) is available during factory and grid search fitting. Examples of data analysis notebooks look like Figures 2, 3, 4.

4. Data and Analysis Preservation

One of the key REP component is the IPython keeper `https://github.com/mikari/ipykee`, or IPykee, which provides comfortable semantic interface to create project and keep tracks of notebooks code along with intermediate results: plots, estimators, data, another objects. All analysis artifacts are saved to the `git` version control system repository. Later anyone who can access to the repository can load project and reproduce experiment or restore its results at any particular point in research. Also, by using nbdiff library, IPykee gives possibility for visual comparison of two versions of one notebook with highilghted differences.

5. Projects using REP

REP has been created as a tool to help in particular data-intensive research projects. Data popularity analysis that combines data on LHCb experiment dataset access for ranking expected popularity of those datasets in the future (`http://github.com/hushchyn-mikhail/DataPopularity`), uniforming (uBoost-like) algorithms A. Rogozhnikov et al. (2015) (`http://github.com/anaderi/lhcb_trigger_ml`), LHCb topological trigger optimization. Search for $\tau \to \mu\mu\mu$ decays at LHCb, anomaly detection of LHC detector and project for search for high-energy cosmic rays — Crayfis `http://crayfis.io`. Our experience shows that tools are not sufficient for making research reproducible, it also required some discipline and good will from team members to follow set of simple practices. With REP those practices just become much easier to follow.

6. Conclusion

Reproducible Experiment Platform is the ongoing project. It provides the environment to conduct reproducible data analysis in a convenient way. It combines different machine learning libraries under uniform well-known interface, meta-algorithms, different parallel cluster interfaces and tools to save and explore intermediate analysis results (code and datasets) and states of analysis. REP was used in several research projects, which list is growing. Source code of the toolkit is available at `https://github.com/yandex/rep`.

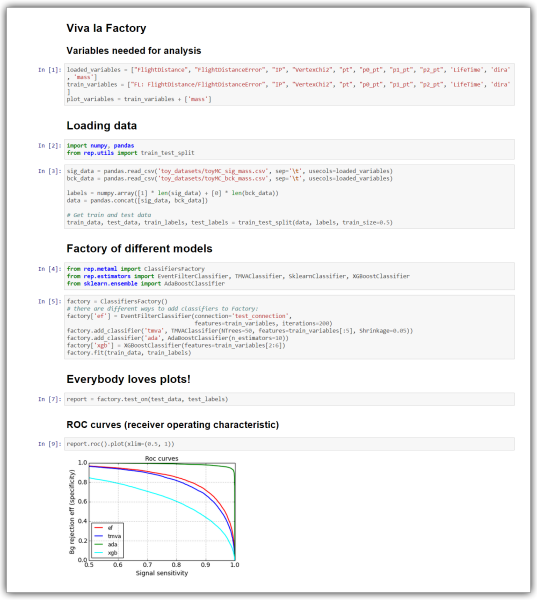

Figure 2: An example of using a factory for training and comparing classifiers.

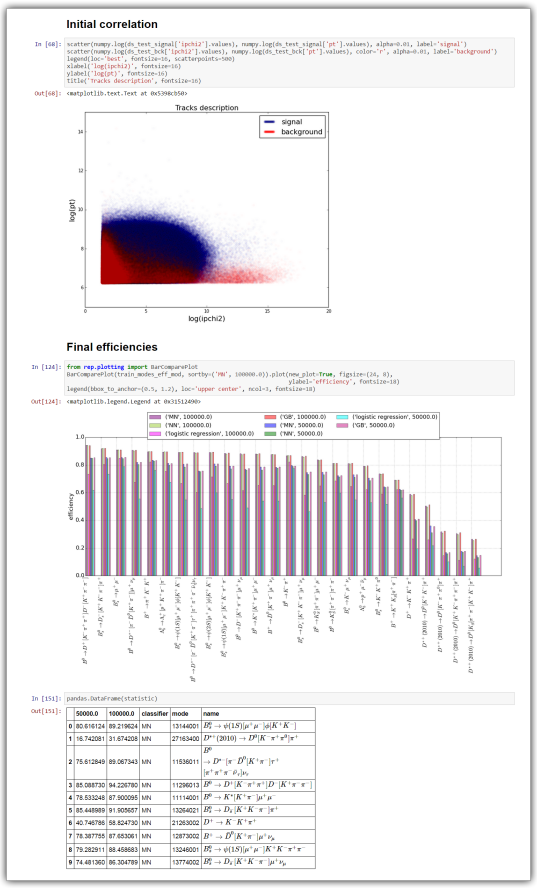

Figure 3: Case-study: part of LHCb topological trigger optimization

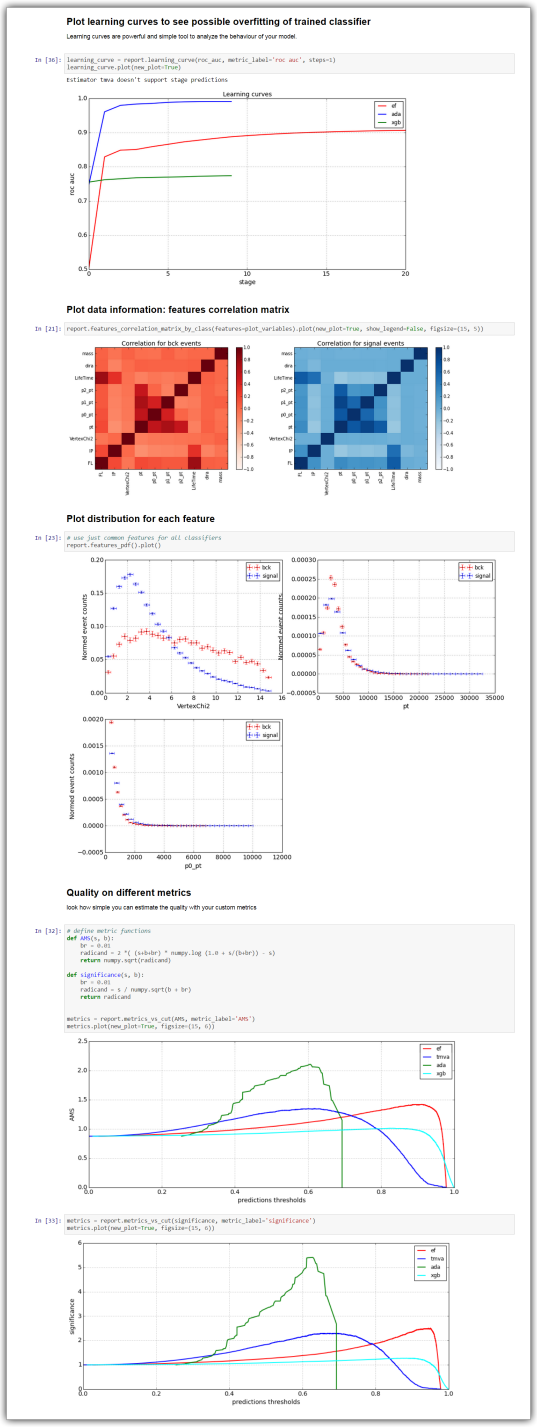

Figure 4: An example of comparing models and plotting results with REP.

References

Fernando Perez, Brian E. Granger, IPython: A System for Interactive Scientific Computing, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007, doi:10.1109/MCSE.2007.53. URL: http://ipython.org

F. Pedregosa et al., Scikit-learn: Machine Learning in Python, 2011, JMLR 12, pp. 2825-2830

S. Chacon, B. Straub, Pro Git, Apress, 2014

A. Hoecker et al., TMVA — Toolkit for Multivariate Data Analysis, 2007, PoS ACAT2007 [arXiv:physics/0703039]

A. Gulin, I. Kuralenok, and D. Pavlov, Winning the transfer learning track of Yahoo's Learning to Rank Challenge with YetiRank, JMLR: Workshop and Conference Proceedings 14 (2011) 63

A. Rogozhnikov, A. Bukva, V. Gligorov, A. Ustyuzhanin, M. Williams, New approaches for boosting to uniformity, 2015 JINST 10 T03002 [arXiv:1410.4140]

T. Likhomanenko, A. Rogozhnikov, A. Baranov, A. Ustyuzhanin, E. Khairullin, Reproducible Experiment Platform, 2015 proceedings of CHEP2015 (to be published)