Automatic validation of big data classifiers on multiple di verse datasets

Automated testing of big data classifiers

Przemysław Czaus

Department of Mathematical Methods of Computer Science, Faculty of Mathematics and Computer Science, University of Warmia and Mazury in Olsztyn, Słoneczna 54, 10-710 Olsztyn, Poland

czaus@matman.uwm.edu.pl

Abstract. While working on data mining applications the main questions are: what do we want to know based on the given data and is the result worth the additional computing power designated for the task. Given the diversity of data and implementations, it is important to select the best-optimized solution for the given task. Every algorithm can behave better or worse when implemented in different languages or even deployed on different architectures. With the expansion of cloud services, distributed programming solutions and containers, optimization even on system level is possible with less effort. The main problem is knowing if the selected solution is better than what was used before. Having the possibility of optimizing the system, algorithm, implementing the solution in a different language or even cleaning the data a different way may give a significant advantage. Most of the published results compare two similar algorithms, on a single machine, written in the same language. The tests differ between scientific manuscripts, sometimes using the same datasets, but without providing the resulting cleansed dataset. That makes the context of the results very narrow and hard to interpret in a bigger scope. The root problem is running unified tests on a variety of solutions and optimization.

Allocation of more resources for the same task sometimes isn't possible and can lead to data loss. Given the scale of some datasets, it is more practical to know if the changes are economically justified. That makes testing some changes on production environments difficult or even impossible. When selecting classifiers, one must first run his own tests using datasets, the same or similar to the production data. Simplifying the process shortens the time from an idea to selecting the best solution for the given job. My main focus is to ensure that every solution is being tested with regard to all of the most important parameters. This way we can measure the impact of changes in the same algorithm as well as the differences between classifiers using the same datasets. Giving a mechanism for standardized tests of new cleansing algorithms, classifiers, language implementations may result in a dynamic progress in a field of data mining. This way one

CEUR-WS.org/Vol-2240/paper6.pdf

can find the best solution and the main differences in a matter of minutes depending on the computing parameters defined for the system.

The main purpose of our study is to build an automated system based on a distributed architecture. Instead of testing the solutions on a single local machine we use a cluster of machines. Those machines can have different hardware. Comparing test results from a base machine with a machine added to the cluster may allow to calculate an accurate difference in processing power so running the same tests on machines with the same architecture shouldn't be needed. For now, the cluster is built on Rancher, that allows deployment of new versions in a matter of seconds. The application consists of loosely coupled modules for dataset storage and cleansing, classifier storage, test generator and automated test runner. Datasets are stored with all their cleansed and test versions. That allows us to monitor the differences between the generated datasets and the result of filtering the data. Tests are generated based on a single cleansed dataset, this way we can see how one data cleaning algorithm impacts the tested implementation. Classifiers are stored in containers, and saved in a container database. The main focus is to ensure every container implements the basic interfaces for communicating with the application (for learning, visualisation and validation). The application communicates with containers asynchronously sending a job to the container and waiting for a response on a designated endpoint. Every classifier has its container version, and new versions are being generated with the given dataset. This way we don't need to teach the classifier when changing any of the system parameters or the processor architecture. Using the container ecosystem gives us the possibility to set some of the system restrictions like a number of processor cores, the size of the memory or even control the number of containers running at the same time. All the containers used for tests are being run in a dedicated cluster. This gives control over the load for the whole system and makes the results more reliable. Adding new architectures and servers to the system is easy and can be done while the application is running. If the whole architecture isn't used some of the servers can be turned off to ensure low maintenance costs of the architecture.

For an example, we want to add a new classifier to the database. We want to test the classifier using already defined standardized tests. First, we have to prepare a container containing a REST API that is implementing basic interfaces used by the application and the classifier that those interfaces send data to. We define what tests to run and we add this job to the queue. If some jobs are currently running we must wait for the processes to finish. The first step is to start the containers and try to teach them with the datasets. If everything is finished we store a snapshot of the container with the data loaded into the database. Every container that finished this process is terminated and the application waits for every container to finish. The next step is to run tests on the previously prepared containers. Every cluster has it's calculated limits and we can run as many copies of the apps at the same time as far as we don't exceed the limit. Every test has its own dataset for learning and for validation. To start the tests we send a package with validation data, this way the test is run locally and we don't have any network delays during the tests. The container measures the time it started

and begins to run the tests. When it finishes it sends the results back to the server, where it is stored and prepared for analysis and publication.

For now, the application can store multiple datasets and their versions with regard to the used cleaning algorithms. One can define default automated tests for new classifier implementations. An advantage for active development is a possibility to build a graph of versions, allowing to analyze what changes generated better results at different datasets. Additionally, we can queue our tests and check the results after everything has been generated.

This solution may be a great way to unify the testing of new classifiers or any algorithms working on cleaning the datasets. Giving everybody a way for fast validation of results of their work. Publishing test results may additionally help many people choose the best solution for a certain task.

Keywords: big data, datasets, classifiers, automated tests