Karlsruher Institut für Technologie

Test Report Document

Numerical Linear Algebra meets Machine Learning

Fabian Koffer

Simon Hanselmann

Yannick Funk

Dennis Leon Grötzinger

Anna Katharina Ricker

Supervisors

Hartwig Anzt Markus Götz

March 15, 2019

Contents

1 Overview

In this last section of our project, we wanted to get our project ready for usage by others. Therefore we first took a look at all the things we wanted to get finished before the project is completed. After we got a list of tasks together, we created issues on GitHub together with a detailed descriptions on what needs to be done. This did not include just bugs, but we also wanted to increase test coverage, remove code issues and set up some necessary things like a Wiki and documentation. After we created these issues, everybody could assign himself the issues he wanted to take care of. While doing so we also discovered some bugs. For these bugs we either created a issue or directly fixed them and wrote them into the bugs-report-table.

2 Statistics

2.1 Code

Lines of Code: 2.852

Python: 2451

C++: 209

others (config and build-setup files): 192

2.2 Testing

Test Coverage: 96%

Number of tests: 63

Number of unit tests: 51

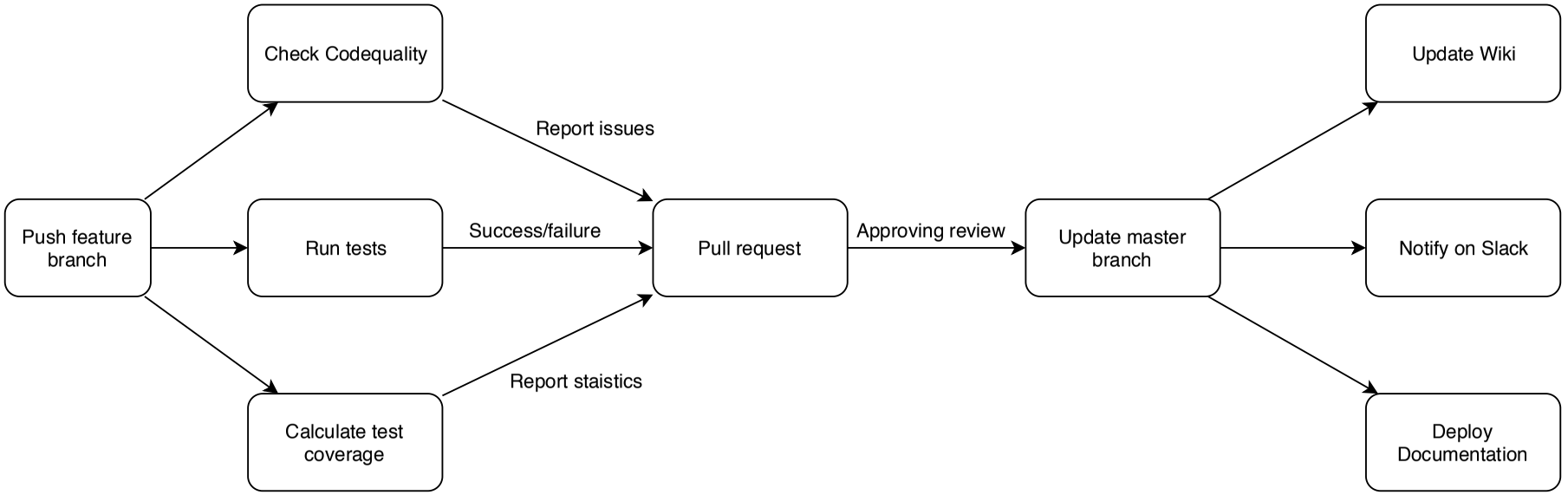
Number of integration tests: 12

3 Continuous Integration

To ensure a high quality on our code, we increased the building process of our code.

When pushing a branch, not only the test are run on Travis, but also is a coverage report generated that is sent CodeClimate. This way you get a small overview on each pull request about the current coverage and the coverage on the newly committed lines. In addition, you can get a complete overview on the CodeClimate Webpage.

We also introduced two new deployment steps that are initiated after a successful merge. The first is that the wiki pages on GitHub are build again in case a change happened on this files. The second step is that we generate a code documentation that is generated and uploaded to GitHub pages.


Check Codequality
Report issues
Push feature branch
Run tests
Success/failure
Pull request
Approving review
Update master branch
Update Wiki
Notify on Slack
Deploy Documentation
Report staistics
Calculate test coverage

4 Code Documentation

In the first phases, we already decided to comment our code with the Doxygen-Syntax. In the implementation phase, we took good care in documenting our code properly. The problem now was that everyone would have to generate this documentation for himself, which is a lot of work.

To simplify this, we added a deployment step, which generates this documentation and deploys it on GitHub pages.

This way, everybody can access the latest version of the documentation by only opening a link.

5 Wiki

Already in the previous phases, we collected some information documenting the workflow we establish in our project. So we decided to make this information available in the GitHub wiki. Because the GitHub wiki holds its own GIT structure and we wanted to keep all our files in one place, we decided to integrate the wiki in our build process.

Therefore we created a folder in the projects root which holds all the wiki's entries in markdown files together with a configuration file. This configuration file is used to dynamically generate a sidebar that holds a navigation for the wiki entries. All the contents of this folder together with the generated sidebar are then copied to the wiki's git.

After this is done, the new wiki entries can be found on GitHub.

6 Bugs

Fault Symptom	Reason	Fix
Command line input with more than one space between arguments resulted in program crash	Input string would be split into a string list which had empty elements that caused problems	Changed the parameters of the string-split function for the expected behavior (remove spaces)
A corrupted configuration file caused the program to crash	Errors that are thrown while opening the file were not caught	Opening config file now happens in a try-except block and errors are reported to user (no crash)
Labeling or collection on operating systems other than linux resulted in crashes	The operating system was not checked when using the labeling or collecting module	Current operating system is checked at start of module and wrong operating systems are reported to user
Changing the size of the collected matrices was not possible	User entered size parameter was not used in collector	Removed static size declaration and started using the size input parameter
Default parameters are not correctly passed to the modules	The configuration file had a wrong format and could not be read properly	The configuration file got restructured and has it's default keys for each module
When not entering anything, the program crashes	The command parser tried to access a element in an empty array	Added extra check to prevent the invalid operation and displaying error message if check fails
Passing a not regular matrix to the classifier causes unexpected behavior	The regularity was not checked on the received matrix in the classifier	We added the regularity check to the classifier and a failure is reported to the screen is this check fails
Trying to open a not existing file in the loader results in a crash	The errors when opening a file where not caught properly	The loader now catches all errors and raises a IOException that can be caught when using the loader
Using the ssget module prints unwanted messages to the command line	The ssget tool has some command line output itself that is printed to the command line	The standard and error output off ssget are now redirected to null so they do not write to standard out
The labeling module was performing unexpected	The accuracy to which the solvers try to solve the matrices was to high	The accuracy was set down to a more realistic value

Fault Symptom	Reason	Fix
Trying to enter a command without setting the name flag results in a program crash	There is no default value for the name	We added a method that sets a default name as a combination of a defaut name from the config file and the current date and time

7 Challenges

Even though we already had some unit tests after the implementation phase, it was not that easy to get the coverage as high as it is now.

While it was quite easy to write tests for the view and the controller, we had a hard time doing the same for the modules. This was because we had a lot of dependencies to other libraries, or even environments, that we could not automate. One example is that we can only use the labeling module on a system where Ginkgo is properly installed. Theoretically, it would be possible to install this in the build process of Travis but our supervisors decided that this would be taking too much time and is not necessary. If we now wanted to test the labeling module, we always had to mock the Ginkgo library. We encountered similar problems with the collector, which has the dependency to the ssget library. This library was easier to set up and was also installed in the build process, so it is possible to write tests against it, but you still end up with more integration tests than unit tests. Another problem concerning the testing was the fact, that most of the modules directly access the memory instead of returning something. This way you could not just assert something of the returning object, but you had to mock the file system and assert calls to this.

Finally the last big problem is the whole structure of machine learning itself. That is, because a big problems of the neural networks is the fact, that you do not have any guarantee on how well it will perform. There is always a big improbability factor in the learning and the classification process. Knowing that, it is difficult to test the process of learning and classification because it will not always perform as expected. You could only assert that the results will be in a certain range, which you would have to define with some heuristics. If you now have this heuristics, the next problem comes up. To get a good result on the training process, you need to use a big data set which will result in very slow test suites which in turn will result in a generally slower workflow.

Together, these aspects make it challenging to use the standard testing and development tools for neural network technology.

8 Glossary

Glossary

CodeClimate A tool that monitors statistics about your code like coverage and displays it on each pull request on GitHub.

Doxygen A tool which uses you comments in the code to generate a documentation of the code.

Ginkgo The ginkgo library is a c++ library which among other things enables an user to solve a linear system with a specified iterative solver and preconditioner. We will be using this library to solve our systems..

GIT A version control system that can be used for tracking changes in a code repository.

GitHub A web based hosting service for the versioning control system GIT.

GitHub pages A web storage hosted by GitHub where you get a personal domain for your GitHub repository.

integration test A test that covers a interaction with the system that contains many modules.

operating system A system software that manages computer hardware and software resources.

ssget A library by the Ginkgo group which lets you download matrices from the suite sparse matrix collection.

Travis A tool for continuous integration that is easy to integrate with GitHub.

unit test A test that only covers one public function and should try to find bugs in this function.