Mini project 3 – Data engineering

Data engineering, 7.5 HP
1TD075

Junjie Chu
Ying Peng
Mandus Hjelm
Xenia Wang

Uppsala Universitet

DECEMBER 10, 2021

1 Introduction

1.1 Problem formulation

In the article *Understanding the Factors that Impact the Popularity of GitHub Repositories* [2] the authors argue that it is of importance for developers to know if their software is attracting new users. Our task is therefor to investigate *which regression model is the best to predict Github stargazers, with repository data from the Github API*. And we set up several VMs and clusters to realize that.

1.2 Data format, Method and Choice of Machine Learning

The dataset where downloaded from Github with their REST API and several features where chosen. When running a cross tabular with Spearmans method the features with an absolute value to close to zero or null where removed. So after that, the features that where chosen are: *forks count, has issues, has pages, has projects, has wiki, open issues count, size of repository, created at day, what day the repository where pushed, description length, repository name length, full name length, login name length, language, length of license string* and the target variable *Stargazers count*. The string features where re-coded or the length of the string where taken and put in to a new feature. Thus, every feature was an int.

We use some built-in functions in scikit-learn for feature selections, such as correlation and the classes in scikit-learn. Based on those, we choose 15 features and extract them from the original data set.

The machine learning methods used where Random forest regression, a simple neural network and Gradient boosting regression. As accuracy measure root squared mean error(RSME) which the model that produced the smallest RSME got pushed in to production server and presented as the prediction model. Both Random forest and Gradient tree boosting are models that combining weak learners, decision trees in to a robust and accurate model[5]. The neural network is built in the framework of Tensorflow, where dense layers combined with dropout to get the highest accuracy and prevent overfitting[1]. These 3 models are chosen because of their differences of methodology and stability from previously known.

When making hyperparameter tuning, we studied the most sensitive parameters of each model. For example, the number of estimators of random forest regression. Therefore, the parameters sets are chosen very carefully to avoid overfitting. At the end, the Ray is deployed for tuning appropriate parameters sets.

2 Framework

2.1 Workflow

The workflow of the project could be described as: first, we extract features , perform feature selection on the development server; second, run the models in different containers on the development server and conduct preliminary model screening; third, using the ray cluster, provide several sets of parameters for the candidate model and perform the tuning; fourth, the best model after parameter tuning is saved in the development server; fifth, on the development server, push the latest model to the production cluster(using githook); last, the user visits the web page, submits a request, and checks the result.

2.2 Architecture

The framework of our project is shown in the figure 1. We have 5 VMs in total: One VM as the orchestration VM, development cluster(several containers in the development VM and a Ray cluster consisting of 2 VMs), production cluster(a Docker Swarm cluster consisting of 2 VMs).

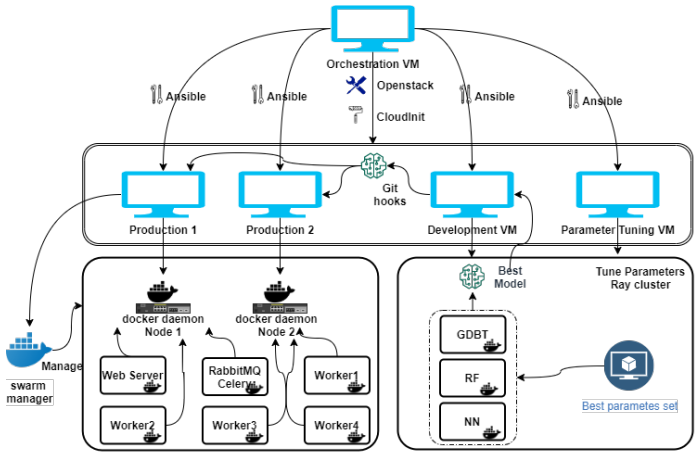


Figure 1: Graph of the architecture

An important part of our framework is the development cluster(Ray cluster). Since the most time-consuming part of machine learning model training is the selection, tuning and adjustment of model parameters, we introduced the Ray cluster. It consists of two parts, a development server(head node) and several parameter servers. We do the feature extraction and selection, model selection in the development server. After we choose the candidate models, we will make use of the Ray cluster to do the parameters selection and evaluate them.

In this project, docker swarm was used to deploy the services in production VMs, which make the whole system more stable and to have a high availability. On the middle right of the figure 1, we deployed development and parameter tuning VMs, we tune the models in Ray Cluster which can speed up the whole process. The best set of parameters will be used to train different models in different containers, and the best model will push by git-hooks from development VM to production1 and production2 VMs.

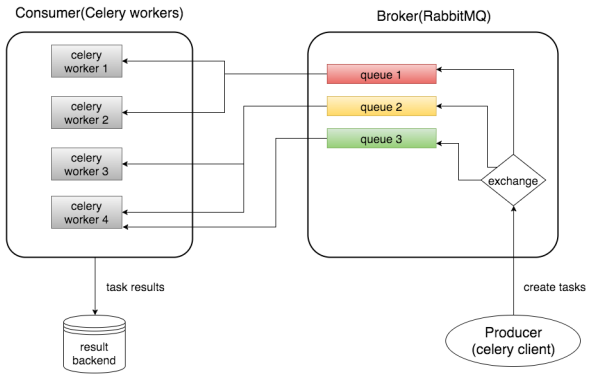


Figure 2: Graph of the Celery and RabbitMQ architecture

On the production server, we run multiple containers. The containers guarantee the isolation between tasks. The tasks running in the container include Flask, RabbitMQ and Celery, as well as workers. Flask is a popular Python web service framework. Our web application is developed based in it. In the worker, we predict the given data based on the model and get the output. Celery is an asynchronous task queue. It can be used for anything that needs to be run asynchronously. The Broker (RabbitMQ) is responsible for the creation of task

queues, dispatching tasks to task queues according to some routing rules, and then delivering tasks from task queues to workers[4]. The framework of RabbitMQ and Celery is shown in the figure 2.

In our case, we submit a request on a flask-based web application. After receiving the request, the celery client calls RabbitMQ to create and assign tasks to workers. After the worker task is completed, the result is stored in the result backend, sent to the web application and displayed.

2.3 Tools, Automation and Orchestration

In order to facilitate the reproduction and management of the project, we have automated and orchestrated most of the steps of the project. Generally speaking, Openstack API is responsible for the automation of creating instances on Openstack and configuring their basic environment, Ansible is responsible for managing and monitoring the VMs, the Ansible playbook is responsible for automatically configuring the detailed operating environment and running programs of each instance, the CI/CD technology(git hook) is used to realize the automatic push of the latest models to the production server. Most of the configurations including ci/cd are orchestrated by Ansible. Jupyternotebook is used to help to edit the code. Part of the construction of Ray cluster and Swarm cluster is also completed automatically by Ansible. But some other settings of the clusters need to be done manually.

Our development cluster is a Ray cluster. It helps us to speed up the training of our models and parameters tuning. We use Docker Swarm to manage our production cluster. It contains 2 VMs. Docker Swarm provides a set of highly available Docker cluster management solutions, which fully supports the standard Docker API, facilitates the management and scheduling of cluster Docker containers, and makes full use of cluster host resources.Both the production cluster and the development cluster have scalability in our architecture.

3 Results

3.1 Scalability Analysis

In this part, we will focus on the test of the scalability of our production cluster. The scalability test of development cluster could be seen in the github. We use Docker Swarm to manage our production server cluster. Through Docker Swarm, we can easily add or replace nodes, and scale containers in the cluster. Therefore, our cluster has horizontal and vertical scalability.

Total requests	Number of workers	Time $/s$	Speedup
1500	1	176	1.0000
1500	2	122	1.4426
1500	4	105	1.6762
1500	8	97	1.8144
1500	12	95	1.8526

Table 1: Strong scalability results

Total requests	Number of workers	Time $/s$
125	1	28
250	2	35
500	4	44
1000	8	81
1500	12	95

Table 2: Weak scalability results

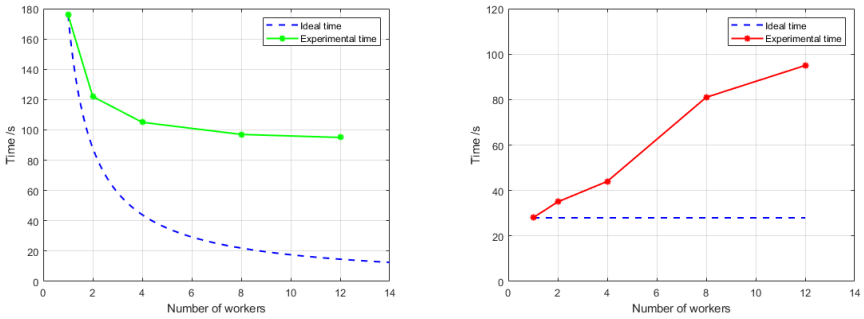
Settings and results of our strong and weak scalability test are shown in the tables above[6]. All the tests are run in the same computer(101.32.242.147, Shenzhen, China). When measuring the time, due to network fluctuations and delays, we choose to use a Python script to send requests of visiting the web in parallel and record the start time, and read the end time in the logs of the workers. The latest end time is regarded as the end time of the entire task. We use the Neutral networks as the model for scalability test since that of the models considered the slowest. We do several experiments with each set of parameters. The median of the elapsed time will be kept, and the time precision is seconds.

The relationship between time spent and the number of workers is shown in the figures below. From the figure, we can see the strong scalability and weak scalability of the cluster. Especially when the number of workers is

greater than the number of CPU cores in the cluster, the scalability is not very good.

Obviously, when the number of workers increases, the overhead for Celery and Rabbit MQ to manage them increases. Moreover, more workers will also bring longer communication time. These time costs are the serial part mentioned in Amandahl's Law and Gustafson's Law [3]. The increase in the percentage of the serial part makes us farther and farther from the ideal situation. On the other hand, when the number of containers (workers) is greater than the number of CPU cores available in the cluster, many tasks are not really running in parallel. Workers need to wait until the CPU core is idle before running tasks. And workers spend more time on one task. For example, at first, one task costs 0.5s, but if we run too many containers, one task costs almost 4s. In this case, our computing resources have not really increased and the occupancy rate of all CPU cores is sometimes 100%. The worst case is that some of the containers crash and are restarted due to hardware performance limitations. So in this case, the scalability of our production cluster is extraordinarily poor.

During the test, we found other problems. We have multiple replicas of workers, but we only have one Web service and Rabbit MQ service. When amount of concurrent visits are large, or we run too many workers, these two services sometimes go down. It will take a lot of time for Swarm to restart them, causing the time of completing the task to be unstable. This is why we choose the median. In the future, we will add more nodes, use high performance hardware and introduce multiple replicas of Web and Rabbit MQ to improve the architecture and enhance the fault tolerance of the cluster.



(a) Strong scalability test

(b) Weak scalability test

Figure 3: Scalability

3.2 Model comparison

Model	RMSE	Best Parameters Set
NN	23151	1st layer:128, 2nd layer:128, 3rd layer:100, 4th layer:40
RF	19260	n_est:90, criterion:mse, max_depth:7, min_samples_leaf:1, min_samples_split:2
GDBT	23730	n_est:100, criterion:mse, max_depth:7, min_samples_leaf:1, min_samples_split:2

Table 3: RMSE of 3 models with best parameters set chosen by Ray tuning

The results we get are shown in the table above. It shows that Random forest is the most suitable model. The advantages of random forest are: it can judge the importance of the 15 features; if a large part of the features are missing(we remove many features at the beginning), the accuracy can still be maintained. It may also have some disadvantages, for example, it is resilient to overfitting even if we can see that its training error is smaller than the Neural network which can be an indicator of that Random forest are more prone to overfit.

The accuracy is not very high and there may be an overfitting problem. To improve them, the size of data set should be increased. And we can try more parameters sets in the future.

4 Collaboration document

4.1 Git repository link

`https://github.com/Junjie-Chu/DE2_Project`

4.2 Contributions

- Junjie Chu:
  I am responsible for setting the clusters(including the Docker Swarm cluster,related python file on the clusters, scalability test, ci/cd and some of the Ansible playbook and debug). And contributed to the report writing and Readme.md.
- Ying Peng:
  I am responsible for setting the clusters(including the Ray cluster, random forest regression code,ray-tune code, related Python files on the clusters, cloud-init files, scalability test, some of the Ansible playbook and debug). And contributed to the report writing and Readme.md.
- Mandus Hjelm:
  I handle the Github API, wrote the downloading script, did the feature selection with help from Xenia, and the model script. And contributed to the report writing.
- Xenia Wang:
  I joined in all meetings and participated to Github API and feature and model selection part, prepared for the slides and contributed to the report writing.

References

[1] Martín Abadi et al. "TensorFlow: A System for Large-Scale Machine Learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[2] Hudson Borges, André C. Hora, and Marco Tulio Valente. "Understanding the Factors that Impact the Popularity of GitHub Repositories". In: *CoRR* abs/1606.04984 (2016). arXiv: 1606.04984. URL: http://arxiv.org/abs/1606.04984.

[3] John L. Gustafson. "Gustafson's Law". In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 819–825. ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_78. URL: https://doi.org/10.1007/978-0-387-09766-4_78.

[4] *Introduction to Celery*. URL: https://docs.celeryproject.org/en/stable/getting-started/introduction.html.

[5] Guolin Ke et al. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3149–3157. ISBN: 9781510860964.

[6] Xin Li. "Scalability: strong and weak scaling". In: *KTH HPC blogs* (2018). URL: https://www.kth.se/blogs/pdc/2018/11/scalability-strong-and-weak-scaling.