No Brains ML

Rohan Jayaram
Thejas Kiran
Vinay Kumar

INTRODUCTION

Traditional machine learning methodologies involve a variety of time-consuming procedures, such as data cleaning, pre-preprocessing, training, and evaluating several models. The **goal** of this project was to automate the iterative tasks involved in creating machine learning models using distributed systems for multiple services like pre-processing, training, and forecasting results.

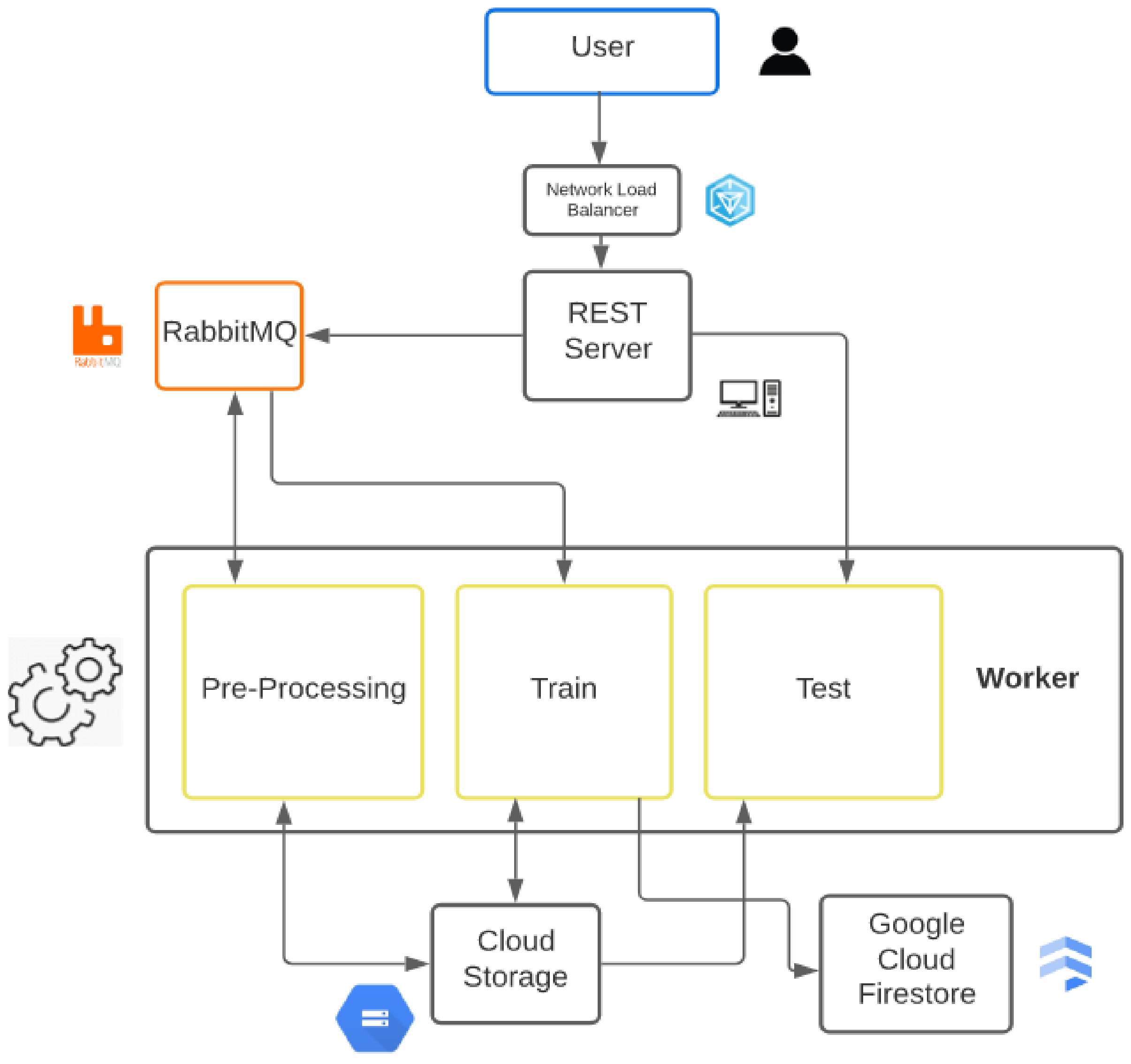
We have **carried out** our project goal by successfully implementing the No Brains ML platform. Our project offers three different web services to automate the whole Machine Learning process. The web services are containerized using Dockers and Kubernetes and use different software components like Google Cloud Firestore and storage, RabbitMQ, and message marshaling techniques for successful implementation.

TECHNICAL AMMUNITION

To achieve our goal, we are planning to use the technologies mentioned below to build our application:

- Google Cloud Firestore
- RabbitMQ
- Flask
- Kubernetes & Docker
- Google Cloud Storage
- REST API
- Message Marshalling/encoding

ARCHITECTURE



INTERACTION

Our implementation design would require us to have a way to distribute user requests by sending them to a network load balancer. Our application can preprocess the whole csv dataset, train on the preprocessed data, provide predictions, and maintain a history of the best models for a given dataset. This is done by sending the user requests to the **REST** Server. Our platform sends the preprocess and train messages to the backend servers using RabbitMQ which then sends the requests to the workers; the test web service directly sends the request to the worker. The CSV (Comma Separated Value) files that are preprocessed by the worker are stored in Google **Cloud Storage** for model building. Once the preprocessing is done, **RabbitMQ** sends a training request to the worker that uses the cleaned CSV file for model building. Currently, the worker tests 2 different regression models with 62 different parameter combinations in total. The best model is then stored in the Google Cloud Storage and the best parameters are also stored in the **Google Cloud Firestore** which is a cloud-hosted NoSQL database. The test worker uses this best stored model for generating predictions to the new dataset. Finally, to manage, scale, and maintain the software, entire application will be containerized and orchestrated on the **Kubernetes** engine.

DEBUG MECHANISM

To ensure that our application runs smoothly and is working as expected, we have used the below mentioned mechanisms to debug -

- We deployed our service on local first to help determine errors and troubleshoot them
- We used port-forwarding for pre-processing and training services
- We used the container logs to monitor deployment

PROS AND CONS OF COMPONENTS

- RabbitMQ: We used RabbitMQ as the messaging protocol for our Platform. RabbitMQ was chosen as it was open-source, and it provides delivery acknowledgement of messages which prevents loss of the messages. The con of RabbitMQ is that it slows down when the dataset is large.
- Google Cloud Firestore: We used Google Cloud Firestore to store the parameters of the best performing model. Firestore was chosen as it is easy to organize and store data and provides a high-performance query engine. The limitation of firestore is that it does not handle complex queries very well.
- Google Cloud Storage: We used Google Cloud Storage to store the best model. Google Cloud Storage was chosen as it is highly efficient and durable and is easy to integrate with Kubernetes. But google cloud can get expensive if resources are not managed well.
- Kubernetes: We used Kubernetes to deploy our application and to manage scaling. Kubernetes was chosen as it integrates well with Google Cloud services and manages several containers simultaneously. But Kubernetes can be hard to deploy and needs a lot of time to deploy.

CAPABILITIES

- Our service is capable of automating tasks such as pre-processing data, model training and model testing which are essential tasks in Machine Learning
- Our service trains with multiple machine learning models and provides the best performing model for the data set

LIMITATIONS

- We have not enabled auto-scaling for services. This might lead to issues when the load is not evenly distributed
- We have used data sets which are minor compared to real-world data sets. Our data set contained about 10000 data points. Larger datasets may lead to high network latency

FUTURE WORK

This application currently can only solve regression problems, I.e., datasets where the target variable is continuous. We have also implemented only 2 machine learning algorithms in this application. The platform can be scaled to solve classification problems in the future. It can also be scaled to perform multiple algorithms to provide the best performing model.

REFERENCES

1. How to Draw Useful Technical Architecture Diagrams
2. Google Cloud Firestore
3. Google Cloud Storage
4. Flask
5. RabbitMQ
6. Kubernetes