WSCBS Assignment4-b Report

Tianhao Xu[1,2[14129027]], Summer Xia[2,3[14094584]], and Yiming Xu[1,2[13284657]]

Vrije Universiteit[1] and Universiteit Van Amsterdam[2]

Abstract. The report gives a brief overview of the Brane framework and the experience we have gained from practice. Through the project on the sinking of the Titanic on Kaggle, which aims to use machine learning tools to predict the survival of passengers. In addition, a method for creating a modular data processing pipeline using Brane framework and the way to automate the testing of our pipeline through Github Actions are provided. We hope to provide readers with some inspiration from our experience, as well as an insight into the strengths and limitations of Brane.

Keywords: Brane · Titanic · Machine learning · Containerization

1 Introduction

Our project topic is Titanic - Machine Learning from Disaster[1]. We are asked to complete an analysis of people who is more likely to survive and use machine learning tools to predict. Also, we are not only focusing on the ML and data processing part, the other goal is to complete the process of this production pipeline through the framework of Brane.

Data Processing Initially, we start from EDA and Feature Engineering to handle the missing value and the useless features. As for the training part, the Decision Tree and Random forest are used to train the model using the processed ten features. The predicted result of Random Forest is higher than Decision Tree with the measurements of metric accuracy and recall.

Data Visualization For visualization, we use Matplotlib and Seaborn with Column Charts, Waveform Charts and Heat map to present the effect of different attributes on survival.

Brane[2] The Brane framework uses containerisation to encapsulate functionality into portable building blocks in the form of workflow applications that perform specified work on multiple nodes distributed across multiple domains. It addresses the challenge of running distributed applications on geographically dispersed IT resources. Scientists can write applications to compose their own data processing pipelines through the framework without having to deal with the underlying technical details. We have divided Tiantic's project into four packages, Initialization, Get Features, Train and Prediction and Visualization, and we composed them into a programmable Brane pipeline.

[1] https://www.kaggle.com/competitions/titanic/overview
[2] https://wiki.enablingpersonalizedinterventions.nl/user-guide/overview.html

The rest of the paper is organized as follows: Section 2 presents the data processing pipeline for Brane. Testing are given in sections 2.4. Section 3 concludes the report with a discussion.

2 Data processing pipeline for Brane

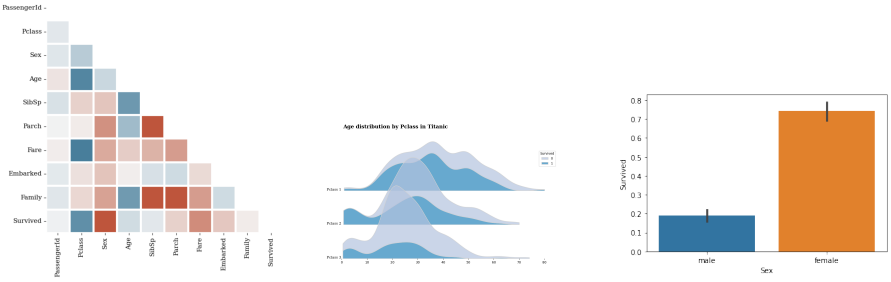
To implement our project within the brane framework, we devided four pipeline components: Initialization, Get Features, train and Prediction, and Visualization.

2.1 Packages

1. **Initialization**[3] We are provided with a shared filesystem by Brane which is created in the */data* directory once Brane is started. The Initialization package is quite different from other packages, it provides a connection between them. It is BraneScript and written by JupyterLab. It loads the raw data from the current directory into the */data* directory first, followed by copying data to the shared filesystem, so that other packages can read it easily.
2. **Get Features**[4] This package performs EDA and Feature Engineering after reading CSV-formatted raw data from the shared filesystem.
   After data investigation, we find that there are 11 attributes. 'Age', 'Cabin' and 'Embarked' variables have a relatively large number of missing values so we need to handle them first. Through the correlation heat map, it is easily to spot the important features on 'Survived' like 'Pclass', 'Cabin', 'Sex' etc..
   For Feature Engineering, we firstly transform a large list of useless name features into sub-typed features to generate some new features. Next, we handle the missing values by filling them with the mean value. But for 'Age', which has many missing values so we construct a **Random Forest** model with 'Sex', 'Title' and 'Pclass' to fill it out with predicted values.
   After EDA and Feature Engineering, the pipeline will generate test & training set and saved to the shared folder.
3. **Train and Prediction**[5] In this package, the training and test data read from *Get Features* package are generated. We utilize the **Random Forest** and **Decision Tree** models to train the model with ten processed features - 'Survived', 'Pclass', 'Sex', 'Age', 'Fare', 'Embarked', 'Title', 'Family', 'Cabin', 'Ticket'. This is the classification task so we choose the metrics like accuracy, recall, F1 and precision score for evaluation.
   After training the model, we make predictions based on the test data. The result in Kaggle shows Random Forest socre(0.78947) is higher than Decision Tree(0.76794). Finally, results are saved to the shared filesystem. Train and test files are been put in the same package because of the limit of memory. The model is also stored as a pickle file in the shared directory.

[3] https://github.com/TISNN/brane-setup
[4] https://github.com/TISNN/brane-getfeatures
[5] https://github.com/TISNN/brane-trainandpredict

4. **Visualization**[6] This package visualizes our findings in the data generated by *Get Features* package. It reads data from the path of shared filesystem. The input provides 5 options including Correlation, Ticket, Title, Gender and Pclass for user to select the figure they want. Figures will be saved to the shared filesystem. The results are partly demonstrated in Figure 4 below.



2.2 Working with submodules

According to the Brane pipeline, we build a GitHub repository for each package and manage these repositories with submodules that allow you to treat a Github repository as a subdirectory of another one.
So we set 'WSCBS_Assignment4b'[7] as the parent repository and the other four 'brane-visualization', 'brane-getfeatures', 'brane-trainandpredict' and 'brane-setup' as subrepositories.
Each of repositories with their own collection of packages. The **'brane import'** command can be used after we setup the repositories. It supports directly downloads a package from GitHub and builds it locally. People may connect to it and download our images from there.

2.3 Create DOI on Zenodo

We can use the data archiving tool Zenodo[8] to archive repositories on GitHub and create DOIs (Digital Object Identifiers) for the archives. This makes the repository easier to reference in the academic literature. After each GitHub repository update, we publish a release again on Zenodo to keep latest. Our final DOI is 10.5281/zenodo.6600849.

2.4 Testing

We created both python unit test and automated testing by GitHub Actions and BraneScript.

[6] https://github.com/TISNN/brane-visualization
[7] https://github.com/TISNN/WSCBS_Assignment4b
[8] https://about.zenodo.org/

1. **Pytest**: Since we are writing each package separately, unit testing for the core functions is necessary to ensure they are executed correctly. So we've built python scripts to test each of our functions individually.
2. **Automated testing by Branescript**: Another complete test is to consider the execution of the pipeline in Brane. For this testing, we created automated test workflow for each Brane package, using GitHub Actions service.
   The steps are as following: I. Setup of Docker, Docker Compose, Docker Buildx. II. Install Brane CLI. III. Build the Brane package. IV. Run package by BraneScript.
   The BraneScript is executed by the *brane run* command. We can determine whether it has successfully completed the task by examining the results of the execution. After accomplishing this, we have actually built the complete CI/CD, which is part of the standard development workflow. Every time we use *git push* to update our code, GitHub Actions will automatically test it based on the workflow we created. For each package in this project, it takes about 6 minutes to complete the BraneScript testing.

3 Discussion

In this project, we built the data processing pipeline with the classification model to predict survival rates using the Brane framework. Thanks to Brane's workflow specification, all packages are implemented as containers that contain all the necessary dependencies for operation. Because of this infrastructure, users can easily implement workflows by calling external functions without needing to know the complex environment and dependencies that these packages are requiring.

But We also encountered many problems while we were using Brane. For a well-developed workflow, Brane can indeed achieve the purpose of easy access for multiple users. However, creating a workflow was not a simple task at the beginning. For users who are unfamiliar with the infrastructure and operation mechanism of Brane, may have troubles with the path of file reading and writing. Once an error occurs, rebuilding the package will take a significant amount of time.

What impressed us the most was to use *repl* to test our package. It showed us the job node failed to create job. We found that the state of the brane-job kept restarting. To solve this problem, we need to create the *infra.yml* and *secrets.yml* files in the config directory.

There still has the limitation for Brane, like packages in the workflow are often closely related. When we update a package in the pipeline, it would effect the upstream and downstream packages which would cause many problems. But We have to say it's an excellent framework. It takes the advantage of docker's simplicity and efficiency in deploying tasks, while also providing an integrated pipeline solution for multiple clusters running the particular tasks. We hope it will become more mature. This project is absolutely a valuable experience. We have learned a lot knowledge and experience of working with cloud-based service and cluster computing from this course. Thanks Adam and all TAs.

4 Contribution

Each team member was extensively involved in the project with balanced workload as table 4.

Member	Work
Summer Xia	1. Worked on Brane packages and wrote report. 2. Brane Scripts Debug & Created testing 3. Finished code documentation.
Tianhao Xu	1. Worked on Visualization and created the package 2. Wrote the report 3. Created DOI 4. Setup the repository and submodules
Yiming Xu	1. Work on the pipeline's construction, including Initialization, Get Features, train and Prediction 2. Construct Brane scripts.

References

1. Barhoom A M, Khalil A J, Abu-Nasser B S, et al. Predicting Titanic Survivors using Artificial Neural Network[J]. International Journal of Academic Engineering Research (IJAER), 2019, 3(9).
2. Ekinci E, Omurca S İ, Acun N. A comparative study on machine learning techniques using Titanic dataset[C]//7th international conference on advanced technologies. 2018: 411-416.
3. Duvall, P.M., Matyas, S., Glover, A.: Continuous integration: improving software quality and reducing risk. Pearson Education (2007)
4. Titanic - Machine Learning from Disaster, https://www.kaggle.com/competitions/titanic/overview. Last accessed 3 Jun 2022
5. Awesome Visualization with Titanic Dataset, https://www.kaggle.com/code/subinium/awesome-visualization-with-titanic-dataset. Last accessed 3 Jun 2022