Data Engineering

Group 6 — https://github.com/DSgroup6/DataEngineeringProject

1 Introduction

In this report, we are going to discuss the Kuzushiji dataset (Feurer, 2019). The Kuzushiji dataset contains handwritten Japanese text. In this project a machine learning model is implemented and deployed into production on the google cloud platform which can recognize handwritten Japanese characters which are given as input. A continuous training pipeline is developed which trains the model using both Logistic Regression and the Decision Tree algorithm, and releases the model with the best performance in the production environment.

There are three main phases in this assignment:

- Building the prediction service- and UI component;
- Building of training pipeline components;
- Executing the pipeline.

The resulting application includes a web application in which the user can input a handwritten character, which is then analysed by the algorithm which returns it's prediction on which character it is to the user. This prediction web application and API are both deployed automatically using a single click on the release trigger in google cloud.

2 Overview of the ML application

The ML application is composed of two components: the prediction UI and the prediction API. The prediction UI is a web application deployed using docker in google cloud, on which the user is able to input their own handwritten Japanese Kuzushiji character by either a file upload or by drawing on a canvas. The prediction API is then contacted by the UI, which makes a prediction on which Kuzusjiji character the user input character is.

Some questions which will be answered are:

- What are the ML Ops requirements for the ML application?
- Retraining requirements (how often does the model need to be retrained?)
- model monitoring requirements (confusion matrix, model performance, evaluation phase)
- model deployment requirements

2.1 Description of the dataset

The dataset Kuzushiji contains 70.000 28 by 28 images of handwritten characters that belong to 10 classes.

If the characters are plotted, we can observe that there are various ways to write a certain character. This can be seen for the first five classes in the below Figure 1.

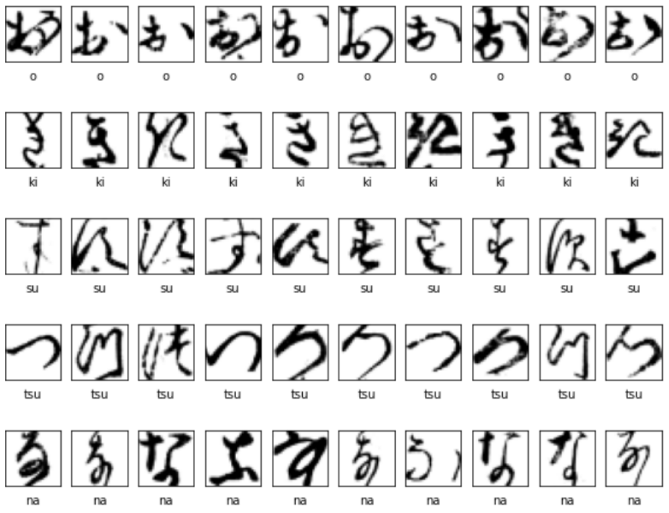

Fig. 1 Representation of the Japanese characters

2.2 Retraining the model

Looking at retraining a model, this refers to updating a deployed model with new data. This can be done in a manual manner or automatically.

Retraining a model has quite some advantages. First of all the content of the data might evolve over time, such as that the handwriting of people change over time. By making use of continuous training the model keeps it's optimal performance on predicting the current type of data. The model is retrained using a automated pipeline, which is executed during each major release of the application.

2.3 Model Monitoring Requirements

To make sure that the deployed model is performing as expected, model monitoring measures are put in place which will enable the maintainers to keep track of any bottlenecks or errors which might occur in the model. First of all logs have been implemented at every section of our pipeline in order to avoid ambiguous errors and tackle problems more easily.

Second of all, a variety of metrics is used to give insight in the quality of the application. Some of these metrics include the

confusion matrix and ROC curve.
In this case in the confusion matrix we are interested in the accuracy, which simply measures the ratio of correctly predicted observations to the total observations. The ROC curve reveals the performance through the creation of a curve representing the TPR (True Positive Rate) and FPR (False Positive Rate): the closer the area under to a 45 degree line, the worst the model performs.

2.4 Model Deployment requirements

Looking at the deployment of the model, this is the phase where the Machine Learning model is moved from an offline environment to an existing production environment.

In our assignment the deployment phase has 4 essential elements:

- Create a model in training environment;
- Cleaning and testing the code;
- Prepare for container development;
- Plan for continuous monitoring and training.

For the first element, which is building the model we have built two classification models, Logistic Regression and Decision Tree. How this has been implemented will be discussed in the next section.

Reflecting on our project, for cleaning and testing the code we did not spend a lot of time on cleaning and testing as we reused a lot of code. Due to this reuse, cleaning and testing the code was not needed.

The third element is preparing the container development. For the container development, we have two containers: prediction UI and prediction API. How these two containers are prepared and implemented will be described in the next section.

3 Design and implementation of the MLOps System

3.1 Description of ML Pipeline

To enable the model to be continuously adjusted and optimised, Continuous Integration in the form of Continuous Learning is applied. A flowchart of the pipeline can be found in Figure 2. In this section, each component of the Continuous Learning pipeline is discussed.

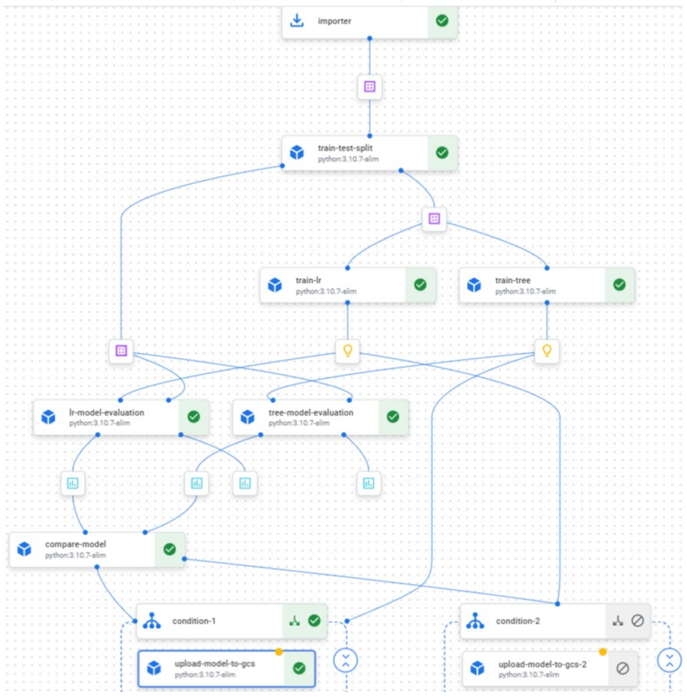

Fig. 2 Representation of the pipeline

1. **Importer:** This component imports the data set from the specified location. It outputs a reference to the data set as an artifact, which is then used for the next component.
2. **Train-test-split:** In this component, we split the data from the importer into a train set and a test set. These are given as outputs in the form of artifacts. The training data is used to train both models used in this report, the test data will be used in later components.
3. **Train-LR:** In this component, we train the Logistic Regression model with the train data received from component 2. The trained model is then given as output in the form of an artifact, which contains a reference to the actual trained model.
4. **Train-tree:** In this component, we train the Decision Tree model on the train data received from component 2. As with the Logistic Regressor, the trained model is saved in a Pickle file and a reference to this file is stored in the output artifact.
5. **LR-model-evaluation:** This component retrieves the reference to the model from component and loads this model. Then, this model is used to predict the test data based on which all sorts of metrics are calculated. These metrics are stored in a AIplatform-experiment and logged in the variables "metrics" and "KPI". These variables are then provided as output in the form of artifacts.
6. **LR-model-evaluation:** This component does everything component 5 does, but then for the Decision Tree model.
7. **Compare-model:** In this component, we compare the evaluations from the previous components. The accuracy of the respective models is stored in the "KPI"-metadata. The model

with the highest accuracy is noted. The return of this component is "LR" in case the Logistic Regressor has the highest accuracy, and "TREE" otherwise.

8. **Condition 1 & 2:** These components decide which model is to be uploaded to the model bucket. Depending on the outcome of component 7, the model is finally uploaded to the model bucket.

The implementation of this pipeline is based on the implementations we were provided during the lab sessions. In particular, lab 4_1 was helpful in providing a solid base for our chosen implementation. The biggest adjustments were needed in the evaluation, comparing and eventual saving of the model. Due to the size of the data set, it was not always possible to use the full test set during model evaluation, due to the fact that the *.json* files storing the data could not be fit into the default sized storage. This inspired us to switch the method of storage from Python dictionaries to artifacts. We will discuss the difficulties we faced in more detail in Chapter 4

3.2 Prediction/serving components

To enable access to the machine learning model a web application is created. This web application consists of 2 components: the prediction UI and the prediction API. The prediction UI is a web application which includes a place where the user can upload an image of a handwritten digit. The UI then requests the Prediction API to make a prediction on which Kazushiji character the user has provided. The Prediction API component uses the machine learning model which is created from the continuous training pipeline to make its prediction. The communication between the components is visualised using the diagram in figure 3.

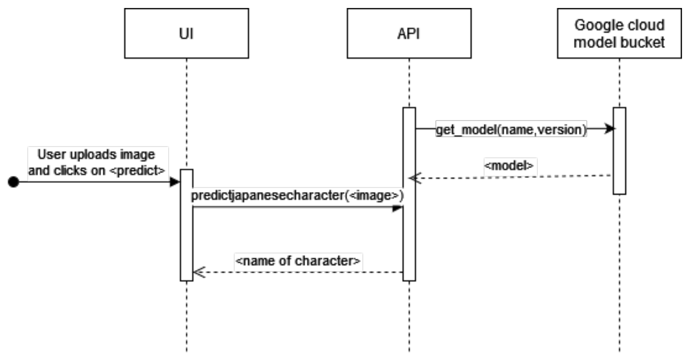
UI
API
Google cloud model bucket
get_model(name,version)
<model>
User uploads image and clicks on <predict>
predictjapanesecharacter(<image>)
<name of character>

Fig. 3 Sequence diagram of the communication between the prediction UI and the API

The prediction web application and API are both deployed automatically in a docker container using a trigger in google cloud. The trigger is executed whenever a release is published to the test/release branch, which enables continuous delivery of the artefacts of both components to the release repository. The purpose of the Test environment is to test if the application is to perform manual tests by stakeholders/testers to test the features of the application. After these tests are complete the release branch is used to release the application to production.

4 Reflection on design and implementation of the MLOps system

In this chapter, we will reflect on the difficulties we faced while designing and implementing the MLops system. Moreover, we will discuss possible alternative methods to what we did.

4.1 Reflection on designing and implementing the pipeline

To start of, we read through and ran the pipeline code provided during the labs multiple times, in order to get a good understanding of what was actually going on. Reading just the code is not enough for this however, as actually designing and implementing your own components gives one a much better picture of what's going on.

The pipeline can be divided into three main phases: the splitting of the data & training the models, evaluating the models and exporting the best model.

The splitting of the data and training of the model did not cause many problems as it is pretty straightforward for an assignment such as this. The same holds true, for the most part, in the evaluation of the model. However, setting up the aiplatform-experiment caused some problems due to the naming of the experiment. Part of our initial evaluation criteria was obtaining the ROC-curve. However, due to the fact that this data set is multi-class we were not able to fully implement this feature. We are able to obtain ROC-curves for each separate class, but not for each class in one figure.

Choosing and exporting the best model required the most attention, as our way of handling the data was different then was described in the lab. In the end, we set up a component which explicitly picks the model with the highest accuracy. This was possible in this case due to the fact that we only have two models. If in the future someone wants to expand on our implementation by adding more models, they will need to rewrite part of this model comparison component.

4.2 Reflection on the deployment of the prediction UI and prediction API

The prediction UI and prediction API pipeline is currently implemented with the idea of creating a Minimal Viable Product (MVP) of the application we have in mind. To make the application ready to be used in production the code needs to be made more robust to to able to be able to handle a larger variety of inputs. One example of this is that the application currently requires the user to upload a 28x28 image. Making this system more robust is important for a production application.

In terms of deployment we currently only use versioning for the code. In the future it would be nice if we also include versioning for the Data and for the Models. By using versioning for data and models, the application can be easily roll-backed to a previous version in case the model performance has decreased over versions. By making use of data versioning, the model can also be trained using the same data as the model was trained with before, which enables us to move back to a previous moment in time.

5 Contribution of the students

In this chapter, we will discuss the contributions each student gave to the assignment. We decided to work on it as a team from the start. Some students were more involved in certain parts than others, as is to be expected from such a project. We worked to the strengths of each member in order to get the project done in time, which means not all students worked an equal amount on all parts.

5.1 Mathieu Janssen 2093815

For this project, I worked on both the code and the report. Initially, we worked collaboratively on the pipeline notebook together. Once the data splitting and training was setup however, I did most of the work for the evaluation and saving of the best model. In the report I wrote the pipeline parts, meaning both the explanation and reflection.

5.2 Nizaar Bechan 2071124

For this project, I was responsible for the structure of the whole report and writing the introduction and part of the overview of the ML application. Furthermore, I made sure I was up to date with all the labs that were shared, so I was able to discuss the pipelines with Mathieu and Luuk and brainstorm about our implementations for this assignment.

5.3 Luuk Ebenau 2092802

For this project, I worked on creating and altering the prediction UI and prediction API. I created the continuous delivery pipeline and triggers, and managed versioning and the division of tech and release branches using our deployment strategy. Besides this I also worked on parts of the continuous learning part in Google Vertex AI, and parts of the report.

5.4 Kheiry Sohooli 1768662

For this project, I worked on the report and wrote a part of section four about the reflection on design and implementation of the MLOps System. Moreover, I was up to date with all the labs so I was able to discuss the topics and implementation methods with Mathieu and Luuk.

5.5 Giorgio Marini 2091737

For this project, I worked on the report and wrote a part of the overview of the ML application. Similar, to Nizaar and Kheiry I was up to date with all the labs and was able to discuss with the team about certain implementations for our project.

References

Feurer, M. (2019, 7). *Kuzushiji-MNIST*. Retrieved from `https://www.openml.org/search?type=datastatus=anyid=41982sort=runs`