# Project Report

## Group 5

Christina Larsson

Ziaul Islam Chowdhury

Suhasini Aswath

# Machine Learning Engineering

# Blekinge Institute of Technology

# Overall System Architecture

The high-level architecture of the machine learning application is shown in the figure below:

A diagram of a model

Description automatically generated

Figure 1: Overall architecture

The architecture of the train delay prediction system is designed in a way so that the training, testing can be continued any number of times without impacting the production backend service. This means that model weights are dumped to files after training and testing phase. The prediction backend service can load the model weights anytime depending on the necessity without any downtime.

It can be seen from the architecture diagram that data is loaded with requested API key from REST API of Trafikverket.se. Response of the REST API is in the format of JSON and contains nested JSON structure. This JSON content is then normalized in Pandas DataFrame. Data preprocessor class does the preprocessing of the normalized content and does feature engineering to make it meaningful for the machine learning model. After completion of data processing, the problem is transformed as a binary classification model with binary label where values are either late or on-time (0/1).

In the next step, Random Forest classifier from Scikit-learn Python package is chosen as the binary classification model. The model is then trained with the dataset and tested. After applying GridSearch, best model parameters are found. The weights of the best classifier is then exported as Pickle file to file storage.

Train delay prediction model loads model weights and initializes Random Forest classifier model. Additionally, label encoders are also initialized as these are required to transform user input coming from the frontend application. In the next step, Flask based web server starts and it offers REST API for the prediction of train delay. When user selects specific train route, date and time, REST service is called to the web server with HTTP POST method. The service then predicts if the train will be delayed or not and returns a JSON data back to the frontend application.

# Data Engineering

Detailed pipeline of data engineering is shown in the following diagram:

A diagram of a process

Description automatically generated

Figure 2: Data engineering pipeline

Figure 2 shows that the TRV REST API is used to get train announcements. API key is required to make REST call and registration is required to get an API key. TRV’s Open API supports various types of data and train announcement is one of them. Data model of train announcement contains the following attributes (description is taken from the schema model):

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| ActivityId | string | Unique ID of the activity. |
| ActivityType | string | Type of the activity (Ankomst/Avgang) |
| Advertised | boolean | Indicates whether the arrival/departure is announced in the timetable. |
| AdvertisedTimeAtLocation | dateTime | Timetable time. |
| AdvertisedTrainIdent | string | Advertised train number (the train number on the ticket). |
| Booking | [] | Code for booking information and booking information, eg: "Carriage 4 unregistered. |
| Canceled | boolean | Indicates whether the arrival/departure is cancelled. |
| Deleted | boolean | Indicates that the data record has been deleted. |
| DepartureDateOTN | dateTime | Expiry date of the Operational train number. |
| Deviation | [] | Any discrepancy with full reason code, e.g.: ABC023 and description, e.g.: "Bus replaces", "Track changed", "Short train", "Not served" etc. |
| EstimatedTimeAtLocation | dateTime | Time of estimated arrival or departure |
| EstimatedTimeIsPreliminary | boolean | Indicates whether an estimated time is preliminary. Note that if the calculated time is preliminary, it means that it can be changed both forward and backward, so a train can, for example, depart earlier than the estimated time if it is also marked as preliminary. |
| FromLocation | [] | From the station of the train in order and in which priority to be displayed. Note that it refers to what is to be advertised to travelers, i.e. what is to be shown on signs etc. In other words, FromLocation can have different content for the same train at different stations and different content for arrivals and departures. The field specifies how from stations are to be advertised. |
| InformationOwner | string | The name of the traffic information owner. |
| LocationDateTimeOTN | dateTime | The operating train's arrival or departure time according to the timetable (may differ from the advertised time). |
| LocationSignature | string | Signature for the station.. |
| MobileWebLink | string | URL to the traffic owner's mobile website. |
| ModifiedTime | dateTime | Time when the data record was changed. |
| NewEquipment | int | Indicates the order in which the train is equipped. If no new equipment has taken place, the value will be zero. |
| OperationalTrainNumber | string | Operational train number (OTN). |
| Operator | string | The railway company that carries out railway traffic, i.e. runs the train, for a traffic organizer. |
| OtherInformation | [] | Code for other advertising information and other advertising information, e.g. "Nice trip!", "Rear vehicle goes locked!", "No boarding". |
| PlannedEstimatedTimeAtLocation | dateTime | Specifies a planned delay and its validity is specified with the PlannedEstimatedTimeAtLocationIsValid flag. |
| PlannedEstimatedTimeAtLocationIsValid | boolean | Indicates whether PlannedEstimatedTime is valid. Will be set to false when an operational estimated time report, time report or slope report is created. |
| ProductInformation | [] | Code for description of the train and description of the train, ex. "Tågkompaniet", "SJ InterCity", "TiB/Tågkomp". |
| ScheduledDepartureDateTime | dateTime | The train's announced departure date |
| Service | [] | Service code and a little extra in addition to product information, e.g. "Bistro", "Sleep and bed". |
| TimeAtLocation | dateTime | When the train has arrived or departed |
| TimeAtLocationWithSeconds | dateTime | When the train has arrived or departed, with seconds. |
| ToLocation | [] | To station for the train in order and in which priority to be displayed. Note that it refers to what is to be advertised to travelers, i.e. what is to be shown on signs etc. In other words, ToLocation can have different content for the same train at different stations and different content for arrivals and departures respectively. The field specifies how to-stations are to be advertised. |
| TrackAtLocation | string | Track |
| TrainComposition | [] | Code for train composition and train composition, ex: "Wagon order 7, 6, 5, 4, 2, 1". |
| TrainOwner | string | The owner of the current train position. |
| TypeOfTraffic | [] | The type of traffic, e.g. "Bus", "Commuter", "Taxi", "Train". |
| WebLink | string | URL to the traffic owner's website. |
| WebLinkName | string | Name of the traffic info owner to use in links. |

Table 1: Data model of train announcement

It’s already mentioned that the response is returned as JSON and contains many fields containing repetitive information. Additionally, some attributes don’t bring any value such as the ID fields or the web links. Hence, data preprocessing is required.

At first, JSON data is converted to Pandas DataFrame and the nested JSON structure is normalized into columns. From the normalized columns, a subset of columns are chosen to be a part of the final dataset. From the column advertised\_time\_at\_location, 9 time features are created such as advertised\_year, advertised\_month, advertised\_day, advertised\_hour, advertised\_minute, advertised\_day\_of\_week, advertised\_is\_weekend, advertised\_season and advertised\_quarter. The categorical features are encoded with label encoding technique.

The final dataset after preprocessing is splitted into training and testing sets with 75% and 25%. With the training and testing sets, grid search algorithm is applied to find best parameters of Random Forest classifier such as maximum depth of the tree. The final Random Forest classifier model is constructed with the hyper parameters. The model is then trained with the training dataset. Finally, testing dataset is applied to the model for prediction and from there confusion matrix and other performance metrics such as F1 score, true positive rate, false positive rate is generated.

As the train delay prediction backend service is separated from the data engineering (training and testing), model weights are then dumped to the file system.

TODO: Add performance metrics (confusion matrix, F1 score)

# Train Delay Prediction Model

Detailed pipeline of train delay prediction model is shown in the following diagram:

A diagram of a train delay classifier

Description automatically generated

Figure 3: Architecture Diagram of Train Delay Prediction Model

Figure 3 shows that the Random Forest Classifier model is initialized from the stored weights along with the label encoders. A Flask server runs on port 8080 and provides 2 REST services to get train schedules and prediction train delay. To assist development and keep track of the APIs, Open API documentation is added with Swagger. Swagger UI lists all available REST services and provides options to execute REST call from the user interface.

Frontend application loads list of available train schedules from the web servers and displays them to the client accessing the application. When a client selects train route along with the departure date and time, /v1/predict-train-delay service is called with HTTP POST method. The prediction service makes a call to the request preprocessor which converts the incoming data to the format which is understood by the machine learning model. The model then makes the prediction if the train will be delayed or not. The response is then returned to the frontend application as JSON format.

# Build Pipeline

To automate the build and deployment of data engineering and train delay prediction system, Jenkins tool was installed on local Docker. A multibranch build pipeline was created and linked to the JenkinsFile located under data engineering part of the project. Idea is to automate build containing pipeline states shown in figure 4.

A screenshot of a computer

Description automatically generated

Figure 4: CI/CD pipeline of data engineering

Unfortunately, the pipeline doesn’t work as expected and it wasn’t possible to fix the Docker related issue (inside of the Docker image of Jenkins) due to time constraints.

# Timesheet diary

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Time Spent | Author | Description |
| Project idea brainstorming | 30 hours | Christina  Ziaul  Suhasini | Discussion on the project ideas. After couple of project ideas and long thinking, finally train delay prediction project was finalized. |
| Meetings with stakeholder | 20 hours | Christina  Suhasini | Various meetings with the stakeholders to understand the requirements. |
| Requirement analysis | 30 hours | Christina  Suhasini | Project object was selected. Various requirements were analyzed. |
| Project architecture design | 30 hours | Ziaul | Designing the project architecture |
| Initial dataset collection | 10 hours | Christina | Collection of train announcement dataset in CSV format |
| Data collection from API and ML model building | 60 hours | Ziaul | Programming data loading, data preprocessing and create Random Forest classifier model. |
| Testing ML model | 20 hours | Ziaul | Testing and optimizing the machine learning model performances. |
| Implementation of train delay prediction service in Python Flask | 50 hours | Ziaul | A web server implemented in Flask (Python) which offers 2 REST services to get train schedules and to predict train delay. User inputs are preprocessed so that the random forest classifier can understand them and make prediction. |
| Testing backend service | 50 hours | Suhasini  Christina | Testing backend service, writing unit test |
| Installing Jenkins and necessary plugins | 20 hours | Ziaul | For the CI/CD pipeline, Jenkins was installed on local machine and necessary plugins such as SonarQube, Pylint, Docker |
| Implementation of CI pipeline | 30 hours | Ziaul | Implementation of various pipeline stages in Jenkinsfile, Dockerfile configuration added as well. |
| Implementation of user interface | 40 hours | Christina | Implementation of user interface in HTML and JavaScript. |
| Writing project report | 100 hours | Ziaul  Chirstina  Suhasini | Preparing architecture diagram, writing introduction, objective, design, technical details, results, conclusions. |
| Preparing project presentation slides | 40 hours | Suhasini  Chirstina | Preparing PowerPoint slides for the presentation. |