Machine learning based decision-support for train traffic disturbance management: An experimental study MATRIX project report

Sai Prashanth Josyula

January 12, 2024

Contents

1 Introduction and background

In Sweden, the railway network is divided into eight control areas, each assigned to a traffic control center (TCC). The railway traffic management is decentralized across these eight centers. During a disturbance, rescheduling the railway traffic is typically handled manually by train dispatchers who have very limited access to decision support systems (Josyula, 2021). At Malmö TCC, timetables are rescheduled using a pen and train graphs printed on paper. At Boden, Stockholm, and Norrköping TCCs, there are decision support systems called STEG and MULTI-STEG. Although these two systems assist train dispatchers in the decision-making process, the conflict resolution is to be performed manually. There are very few examples of real-world railway traffic management systems that can compute and suggest rescheduling decisions to the train dispatchers during disturbances (Josyula, 2021).

During railway disturbances, the time available for analyzing alternative rescheduling decisions is often very limited, e.g., one minute. In Sweden, as per Trafikverket's guidelines, trains that are running as per their planned schedule are prioritized at the respective infrastructure location. Therefore, during a disturbance, a frequently employed rescheduling strategy is to prioritize the on-time trains over other trains. However, this strategy does not always lead to the best rescheduling solution as several potentially desirable alternative schedules are never considered. It is a challenge for the decision maker to analyze alternative desirable solutions and motivate their rescheduling choices within the available time (Josyula, 2021).

The need for fast and intelligent decision support systems for rescheduling railway traffic is evident. While designing and implementing such fast intelligent decision support systems, the use of artificial intelligence (AI) approaches is often inevitable and offers significant benefits. In the MATRIX project, we investigate machine learning (ML) based decision support for train traffic control in case of disturbances through an experimental study.

2 Problem description and scope

When a delay occurs in a railway network, train dispatchers need to detect potential conflicts and resolve them so that the propagation of delay to other trains across the network is mitigated (Wen et al., 2019). In Sweden, a typical strategy used by the dispatchers when resolving a conflict is to prioritize an on-time train over an already delayed train. One reason for adopting this prioritization strategy is due to the lack of practical decision support tools that show alternative rescheduling solutions obtained from different prioritization decisions in real time.

In the context of railway disturbance management, a train timetable rescheduling algorithm is an important decision support tool with three main goals (Obara et al., 2018): (i) to quickly reach completion, (ii) to capably handle large, realistic input data, and (iii) to obtain high-quality solutions. The actions performed by a typical rescheduling algorithm can be broadly categorized into two main tasks: (i) computing alternative rescheduling solutions and (ii) selecting a solution based on the objective(s). The computation of alternative solutions primarily involves employing different rescheduling tactics to resolve the identified potential conflicts. We only consider the following three tactics during rescheduling: (i) retiming, i.e., allocating new arrival and departures times to one or more trains, (ii) local rerouting, i.e., allocating alternative tracks to one or more trains, and (iii) reordering, i.e., prioritizing a train over another. The focus of this study is to effectively determine the train to be prioritized for any potential conflict.

An aim of our study is to explore the benefits and challenges of developing ML-based approaches to reschedule train traffic whenever potential conflicts are identified. In this study, we devise an ML model that suggests with certain confidence which train is to be prioritized during a conflict. The study's scope is limited to predicting/suggesting the best prioritization alternative and excludes predicting other variables in the decision-making process, e.g., whether to change the track or retime a train that is in a potential conflict. Predicting the best prioritization alternative is expected to be sufficient and rewarding as the devised ML model is intended to work alongside an existing parallel algorithm for train rescheduling. The ML model can interact with the rescheduling algorithm as well as the train dispatcher, who is the human decision maker. The dispatcher is ultimately responsible for deciding the timetable adjustments that will lead to a rescheduled timetable.

Our work focuses on the infrastructure manager (IM)'s need to manage the railway traffic and revise the train timetable during disturbances. It hence excludes disruptions and rescheduling of track maintenance. We also do not explicitly consider the effects of the disturbance on the rolling stock and crew schedules.

The aim of our research is to create an ML model that can assist train dispatchers in resolving potential conflicts in a train timetable. The main goals of this research are: (i) to explore the opportunities that state-of-the-art ML algorithms may offer the train traffic domain and (ii) to provide valuable insights on incorporating ML when rescheduling train traffic during disturbances. The main objectives are: (i) to model the conflict resolution in a timetable using ML and artificial training data, (ii) to devise an ML model for data-driven decision making to resolve potential conflicts in a train timetable during rescheduling, and to assess the model's feasibility and effectiveness.

3 Related work

Solution approaches frequently used by researchers to solve rescheduling problems include mathematical models and exact or heuristic algorithms. Although the potential of mathematical models is proven in many research studies, their use for real-world rescheduling may be difficult as they typically cannot incorporate train dispatcher's knowledge and expertise (Wen et al., 2019). On the other hand, data-driven solution approaches can capture the dispatchers' experience and lead to more practical rescheduling decisions. An example of data-driven approaches are ML approaches, which are receiving remarkable attention in recent years for solving scheduling problems (Fazel Zarandi et al., 2020; Karimi-Mamaghan et al., 2022; Li et al., 2021). They have been shown to work well in transportation and many other problem domains, independently or in combination with traditional approaches, to tackle complex (re)scheduling problems and get better solutions (Li et al., 2021; Karimi-Mamaghan et al., 2022). For a comprehensive review of data-driven approaches for train dispatching management, see Wen et al. (2019).

Recently, Tang et al. (2022) present a systematic literature review of AI applications in railway systems, wherein they identified nine research studies that use AI for railway rescheduling. Only few of the nine studies used ML, some of which are as follows. Obara et al. (2018) use a deep reinforcement learning (DRL) approach which effectively reschedules trains during disturbances caused due to train delays (of 5–35 min) on a small-scale problem consisting of six trains and eight stations. Ning et al. (2019) present an approach to rescheduling high-speed trains by using DRL to adjust running and dwell times and reorder trains at stations. Using their approach, the total train delay in the resulting rescheduled timetables could be reduced significantly compared to a first-come-first-served approach. Kuppusamy et al. (2020) use deep learning (DL) along with an improved genetic algorithm to optimize the energy of metro systems during rescheduling. See the

recent literature review by Zhang et al. (2023) for discussions on the application of AI for railway traffic management from various perspectives.

The above research studies used deep and reinforcement learning approaches for rescheduling. In contrast, the use of supervised ML approaches for train timetable rescheduling is rarely investigated. Supervised ML approaches can be used during train timetable rescheduling in many ways. For example, Li et al. (2022) modelled the train conflict detection as a classification problem. They used information from the current and past stations to detect potential conflicts at the next station. In their study, the authors trained a Bernoulli Naive Bayes model using 12 features, of which two features are infrastructure-related, three features are weather-related, and the remaining seven features are timetable-related. The size of their dataset is 81,707 examples, of which 70% are used for training the model and tuning its hyperparameters while 30% are used to test the model. Only few research studies use ML-based approaches for conflict resolution in railway traffic management. In one of the early works, Dündar and Şahin (2013) devise a supervised ML approach, using artificial neural networks (ANN), to resolve a conflict between two trains on a single-track network. The authors used a dataset containing 331 conflicts resolved by dispatchers over ten days. An ANN classifier was trained with 52% of the data, while the remaining data was used for validation and testing the model. The following four features were used for the two trains in conflict: train priority (based on its speed class), a measure of lateness, conflict resolution delay, and number of potential conflicts. Their ANN acted as a binary classifier that determined which train passes and which train stops while resolving a potential conflict.

ML-based approaches for conflict resolution in railway traffic management are relatively limited, e.g., compared to air traffic control. In a recent literature review, Wang et al. (2022) present 19 primary studies that use DRL approaches for conflict resolution in air traffic control. A recent study by Rahman et al. (2022) presents a multi-label classification model that suggests different heading directions to air traffic controllers to avoid aircraft conflicts. To the best of our knowledge, research on investigating similar classification models in the context of train timetable rescheduling is very limited. The aim of this study is to investigate the use of supervised ML algorithms and create classification models that have *learnt* to effectively prioritize trains when resolving potential conflicts in a train timetable. The end goal is to support train dispatchers during the conflict resolution process of train timetable rescheduling.

4 Problem formulation

Given a potential conflict between two trains that arises in a train timetable over a railway infrastructure, the goal of our ML model is to recommend to the dispatcher the train to be prioritized. The model's output is a label, which is the name of the train to be prioritized when resolving the input potential conflict. The role of the ML model is to make data-driven prioritization suggestions. Following the recommendations of the ML model to resolve every potential conflict in a disturbed timetable results in a rescheduled timetable.

To learn a model that suggests the train to be prioritized during a potential conflict, we create a dataset D comprising potential conflicts and associated prioritization decisions using a set of rescheduled timetables. Each data row D_i in the dataset comprises a feature vector x_i and the corresponding output label y_i. We use a train timetable on the Karlskrona–Hässleholm–Malmö stretch, simulated disturbance scenarios, and their rescheduled timetables obtained using an existing heuristic rescheduling algorithm that aims to minimize the total final delays greater than 3 min.

We assume that the train prioritization decisions in the rescheduled timetables used to create our dataset D are preferred by decision makers over the alternative decisions.

Two solution approaches for solving the above problem are *one-versus-one* (OvO) strategy and *one-versus-rest* (OvR) strategy. In OvO strategy, a binary classifier is created for each pair of classes. For example, when there are N classes, this strategy creates $N(N-1)/2$ classifiers. An advantage of this strategy is that each binary classifier needs to be trained only on a subset of the training set (Géron, 2022). In OvR strategy, a binary classifier is created for each class, thus creating N classifiers for N classes.

5 Machine learning pipeline

The infrastructure surrounding an ML algorithm is called a *pipeline* (Google, 2017). An ML pipeline includes data collection, creating training data files from the collected data, training alternative models, and exporting the models to production (Google, 2017). A pipeline can also be defined as a sequential combination of various algorithms that transforms a dataset into a ML model (Zöller and Huber, 2021; Burkov, 2020). According to Zöller and Huber (2021), a good ML pipeline contains "specialized data preprocessing, domain-driven meaningful feature engineering, and fine-tuned models resulting in high predictive power". Building a good ML pipeline requires a deep knowledge of ML algorithms and the problem domain and is typically a complex task, performed iteratively using trial-and-error (Zöller and Huber, 2021). Automated machine learning (AutoML) is the task of automating the creation of a pipeline for the problem at hand (Wever et al., 2021; He et al., 2021).

The first part of our pipeline involves feature engineering (see Figure 1) while the rest of the pipeline was planned to be obtained using a genetic programming-based pipeline optimization tool (Olson and Moore, 2016). The pipeline output by the tool may comprise various algorithms, but most importantly will include an ML algorithm with tuned hyperparmaters. The hyperparameters, their type (e.g., binary, categorical, continuous, discrete), and their tuning process varies on the ML algorithm. The hyperparameter configuration of an ML algorithm has a direct impact on the resulting model's performance (Yang and Shami, 2020).

When we used the aforementioned pipeline optimization tool, called *TPOT*, on the *Blekinge Kustbana med omnejd T21* dataset (created as part of the MATRIX project), it did not result in ML models with a high accuracy even after investigating thousands of pipelines for many hours. On the other hand, manually investigating a few different algorithms gave us accurate ML models. Hence, we do not present results from the pipeline optimization tool in this report and create the second part of the pipeline in Figure 1 manually in this study.

Figure 2 shows how and where the ML model created using the above methods fits in the context of decision support for train traffic control during disturbances. As can be seen in Figure 2, a rescheduling algorithm (Josyula, 2021) will work alongside the ML model, and suggest specific conflict resolution decisions (e.g., make Train i wait t seconds at Section j) and a rescheduled timetable to the train dispatcher. In the future, as a dispatcher interacts with the model, it can 'continuously' learn to capture their preferences.

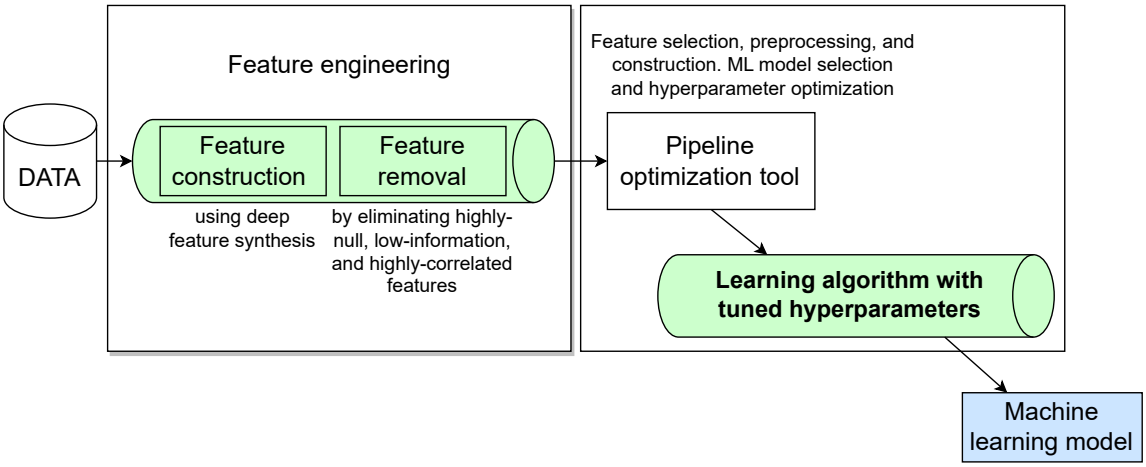


Figure 1: The initially designed machine learning pipeline for this study

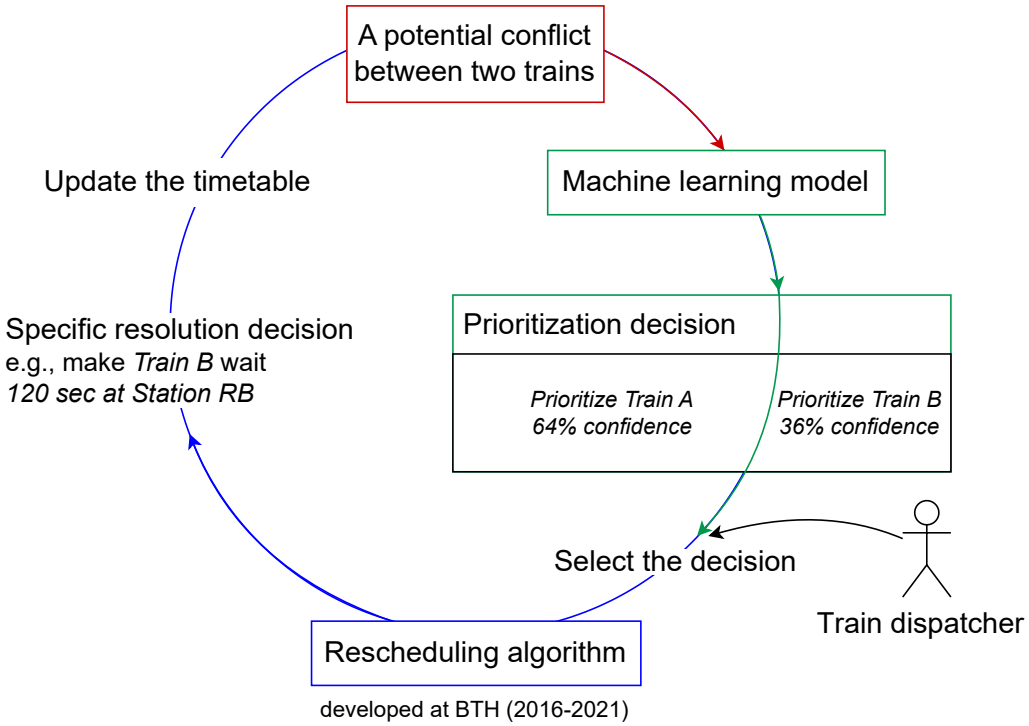


Figure 2: How the machine learning model can be used during rescheduling

6 Feature engineering

Feature engineering is the process of transforming raw data into features that can be used by ML algorithms. It encompasses many different techniques, e.g., constructing, selecting, and extracting meaningful features. Feature construction is the process of constructing new features from basic features or raw data to enhance the model's performance (He et al., 2021). Feature selection is the process of selecting a subset of features by removing irrelevant features (He et al., 2021). Feature engineering is an important step that has a significant effect on the resulting model's performance.

We use various feature engineering techniques, but primarily a method called *deep feature synthesis*, to synthesize meaningful features that may impact which train should be prioritized. A potential conflict that arises due to a disturbance has the following notable features: (i) the section where the potential conflict exists, (ii) the two trains causing a potential conflict, their entry and exit times on the conflict section, and the track occupied by them. In the following section, we discover more features using deep feature synthesis.

6.1 Deep feature synthesis

Deep feature synthesis (Kanter and Veeramachaneni, 2015) is an algorithm that can be used to generate features for relational datasets. The goal of the algorithm is to extract relevant features from the dataset based on relationships between different *entities* (data tables). Figure 3 gives an overview of the six tables comprising our raw data and presents the following details: (i) the number of rows in each data table, (ii) the names of the columns and the type of data in each column, and (iii) the relationships between various data tables.

Deep feature synthesis differentiates two kinds of relationships: *forward* and *backward* (Kanter and Veeramachaneni, 2015). The direction of the arrows in Figure 3 determines whether two data tables have a forward or a backward relationship. A forward relationship is between a row of table T^l, and exactly one row of another table T^k (Kanter and Veeramachaneni, 2015). The *conflicts* table has a forward relationship with the *Infra* table (*conflicts* $\rightarrow$ *Infra*). The reason is that each conflict in the *conflicts* table can be mapped onto exactly one infrastructure section. A backward relationship is a relation from a row in a table T^k to all the rows in a table T^l that have a forward relationship with it (Kanter and Veeramachaneni, 2015). In the above example, the *Infra* table has a backward relationship with the *conflicts* table (*Infra* $\leftarrow$ *conflicts*). The reason is that an infrastructure section can be mapped onto several conflicts. Table 1 summarizes the four relationships that the *Infra* table has with other data tables shown in Figure 3.

Deep feature synthesis generates features called direct features, relational features, and entity features, denoted by `dfeat`, `rfeat`, and `efeat` respectively, based on the relationships between the entities. Direct features are applied over the forward relationships while relational features are applied over backward relationships. For more details on deep feature synthesis, see (Kanter and Veeramachaneni, 2015).

6.2 Synthesized features

The data used for feature engineering presented in this document is from the *Blekinge Kustbana med omnejd T21* dataset, which was created during the MATRIX project. The *conflicts* data table consists of 194,863 conflicts and prioritization decisions across 18,000 simulated disturbance scenarios.

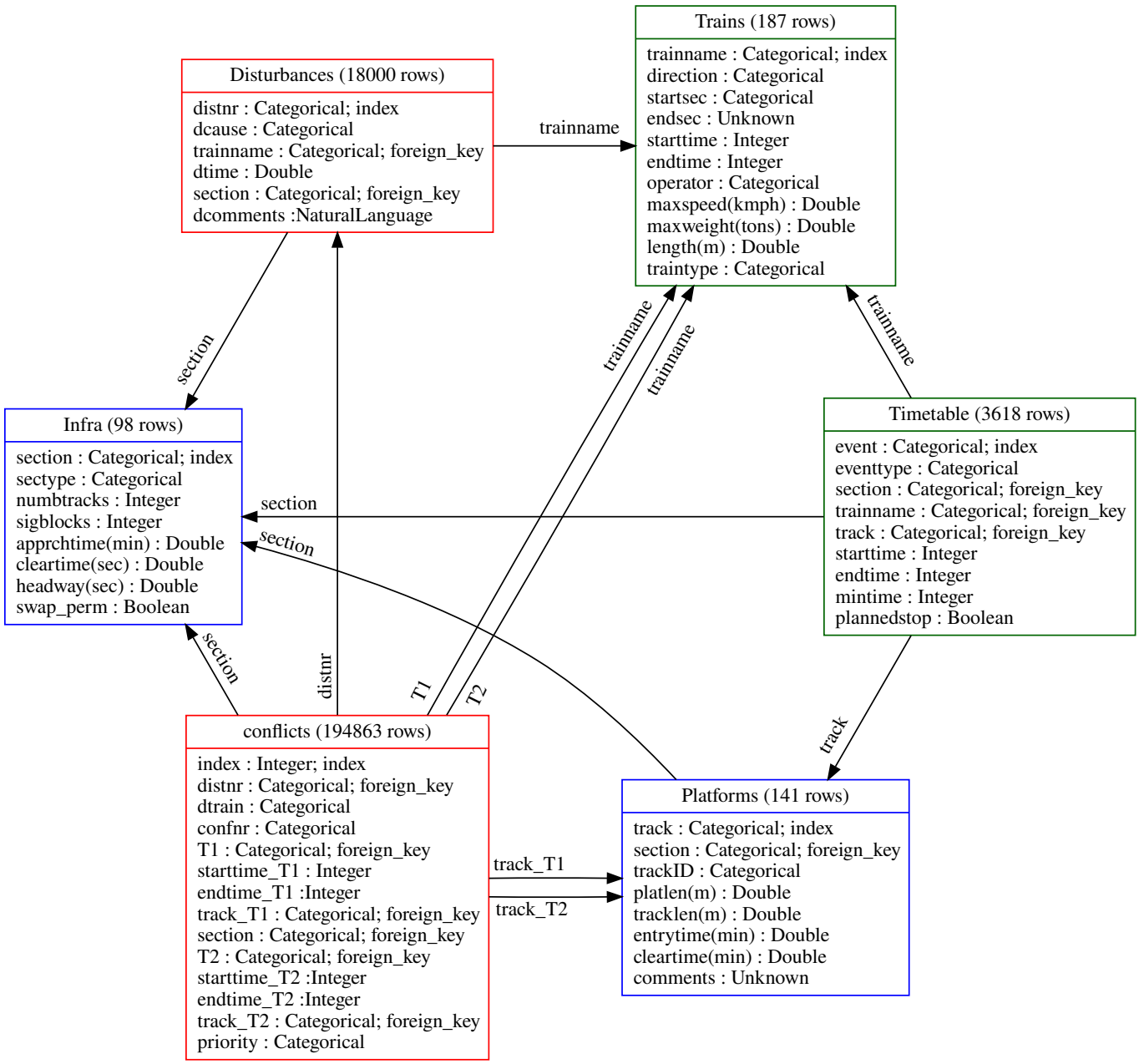


Figure 3: Relationships between different data tables and the data types

Table 1: Examples of relationships that *Infra* table has with other data tables.

Backward relation with	Reasoning behind the relationship
conflicts	A section can have many potential conflicts over it; each row in *conflicts* contains details about a potential conflict.
Disturbances	A section of the infrastructure can be a disturbance section more than one time in the dataset; each row in *Disturbances* contains details about a disturbance scenario.
Timetable	A section can have many timetable events scheduled on it; each row in *Timetable* contains details about a train event.
Platforms	A section can have many platforms; each row in *Platforms* contains details about a platform.

Table 2: Three rows of the *conflicts* table

(a) Columns 1–9 of the data table

index	distnr	dtrain	confnr	T1	starttime_T1	endtime_T1	track_T1	section
0	D1	491	C1	491	57493	57961	MGB-M#2	MGB-M
1	D2	491	C1	491	57493	58261	MGB-M#2	MGB-M
2	D3	491	C1	491	57493	58561	MGB-M#2	MGB-M

(b) Columns 10–14 of the data table

T2	starttime_T2	endtime_T2	track_T2	priority
1277	57658	57823	MGB-M#2	491
1277	57658	57823	MGB-M#2	1277
1277	57658	57823	MGB-M#2	1277

The columns of the *conflicts* data table are shown in Table 2. The three disturbances shown in Table 2 (D1, D2, and D3) created only one potential conflict each. However, note that a disturbance typically creates many potential conflicts. The target variable is `priority`, which holds the name of the train that is prioritized when resolving the conflict. This is the variable that the ML model will predict. Figure 4 presents an overview of the steps taken during feature engineering. Based on the relationships between various data tables of Figure 3, 442 features are synthesized in total using the deep feature synthesis algorithm. In the feature matrix, each row corresponds to a potential conflict's data and each column corresponds to a feature. From the 442 features, 202 were removed as more than 90% of values in the corresponding columns of the feature matrix were null. Of the resulting 240 features, 208 features with at least two unique values were retained while the remaining were rejected as *low information features*. Of these 208 features, 88 correlated features were identified using the Pearson correlation coefficient and removed, resulting in a set of 120 features. Of the 120 features, 25 duplicates were removed to arrive at a set of 95 features. From these 95 features, we manually removed 32 features and arrived at 63 features.

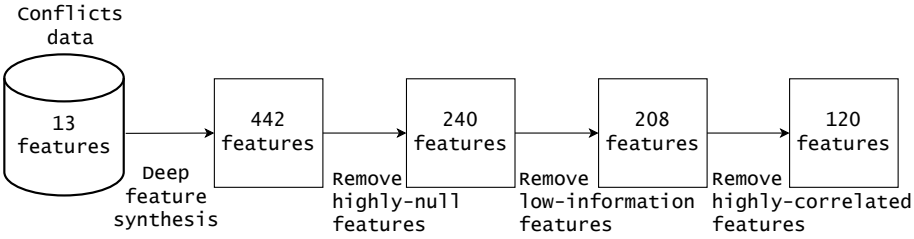


Figure 4: An overview of the performed feature engineering process

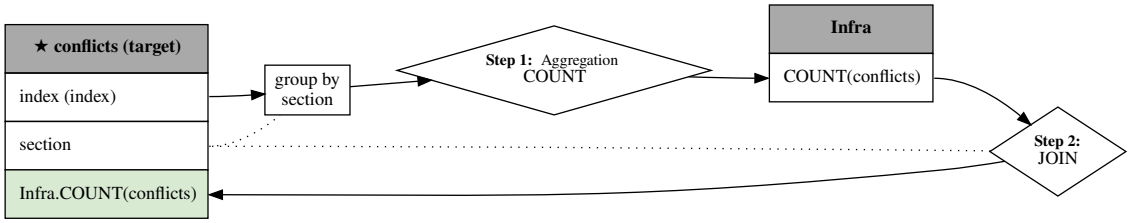
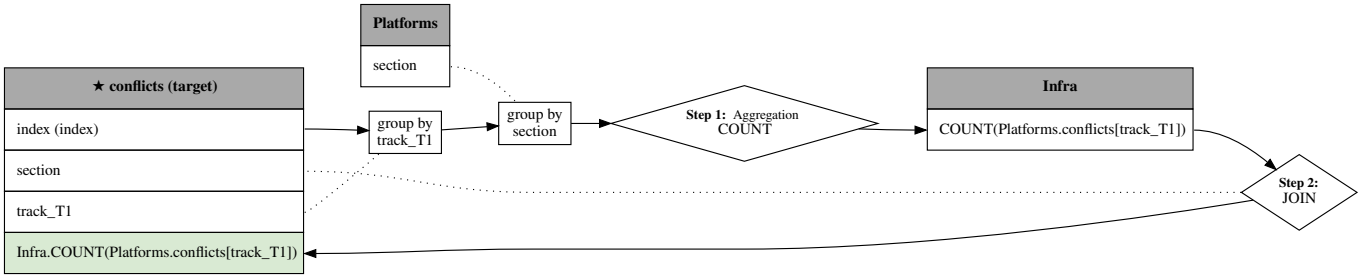
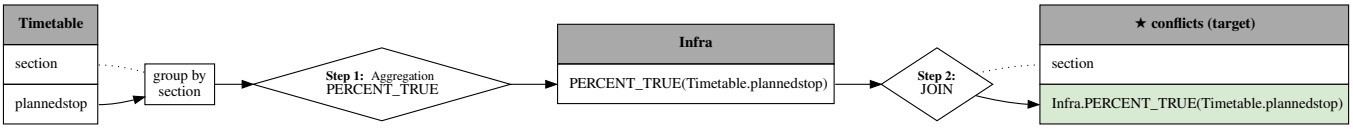


Figure 5: A *depth-1 feature*, `Infra.COUNT(conflicts)`: The number of times conflicts have occurred at the section

Figure 6: An example of a depth-1 feature in the synthesized features



(a) `Infra.COUNT(Platforms.conflicts[track_T1])`: The number of potential conflicts at track_T1 if it has a platform, else 0



(b) `Infra.PERCENT_TRUE(Timetable.plannedstop)`: The percentage of trains in the timetable with a planned stop at the conflict section.

Figure 7: Two examples of deeper features in the synthesized features

Table 3: A few synthesized features (35 out of 63) and their description

Feature	Description
Conflict-specific features	
section	The name of the conflict section C
T1	The train in the potential conflict that reaches C first
starttime_T1	The start time of the first train at C
endtime_T1	The end time of the first train at C
track_T1	The track of the first train at C
T2	The train in the potential conflict that reaches C second
starttime_T2	The start time of the second train at C
endtime_T2	The end time of the second train at C
Infra.sectype	Type of the conflict section
Infra.numbtracks	Number of tracks at C
Infra.sigblocks	Number of signal blocks at C
Disturbance-specific features	
Disturbances.trainname	Train originally disturbed due to the disturbance
Disturbances.dtime	The primary delay (in minutes)
Disturbances.section	The name of the disturbance section
Conflict dataset features	
Infra.COUNT(conflicts)	The number of times conflicts have occurred at C
Infra.MODE(conflicts.T1)	The train that most frequently involves in potential conflicts as T1 at C
Infra.MODE(conflicts.T2)	The train that most frequently involves in potential conflicts as T2 at C
Infra.MODE(conflicts.track_T1)	The track that most frequently involves in potential conflicts as track_T1 at C
Infra.NUM_UNIQUE(conflicts.track_T1)	The number of unique tracks involved in potential conflicts at C
Disturbance dataset features	
Infra.MODE(Disturbances.trainname)	The train that most frequently experiences a primary delay at C
Original timetable features	
Infra.MAX(Timetable.endtime)	The end time of the train that leaves the conflict section last
Infra.MAX(Timetable.mintime)	The maximum of the mintimes of train events using C
Infra.MEAN(Timetable.endtime)	The average of the end times of all the trains using C
Infra.MEAN(Timetable.mintime)	The average of the min times of all the trains using C
Infra.MIN(Timetable.endtime)	The end time of the train that first leaves C
Infra.MODE(Timetable.track)	The most frequently occupied track at C as per the original timetable
Infra.MODE(Timetable.trainname)	The train that traverses C most number of times. Note: each train passes each section only once in our timetable
Infra.PERCENT_TRUE(Timetable.plannedstop)	The percentage of train events with a planned stop at C
Infra.SKEW(Timetable.mintime)	The skewness of min times of trains passing C
Infra.STD(Timetable.endtime)	The standard deviation of end times of trains passing C
Infra.SUM(Timetable.mintime)	The sum of minimum occupancy times taken by trains at C
Conflict section features	
Infra.SUM(Platforms.cleartime(min))	The sum of cleartimes of all platforms of the conflict section
Infra.COUNT(Platforms.conflicts[track_T1])	The number of potential conflicts at track_T1 if it has a platform, else 0
Disturbance-specific features	
Disturbances.Infra.numbtracks	The number of tracks at the disturbance section
Disturbances.Infra.sigblocks	The number of signal blocks at the disturbance section

The 63 features can be categorized based on the *depth* used by the deep feature synthesis algorithm to create them. Figures 6 and 7 show examples of features of different depths, and how

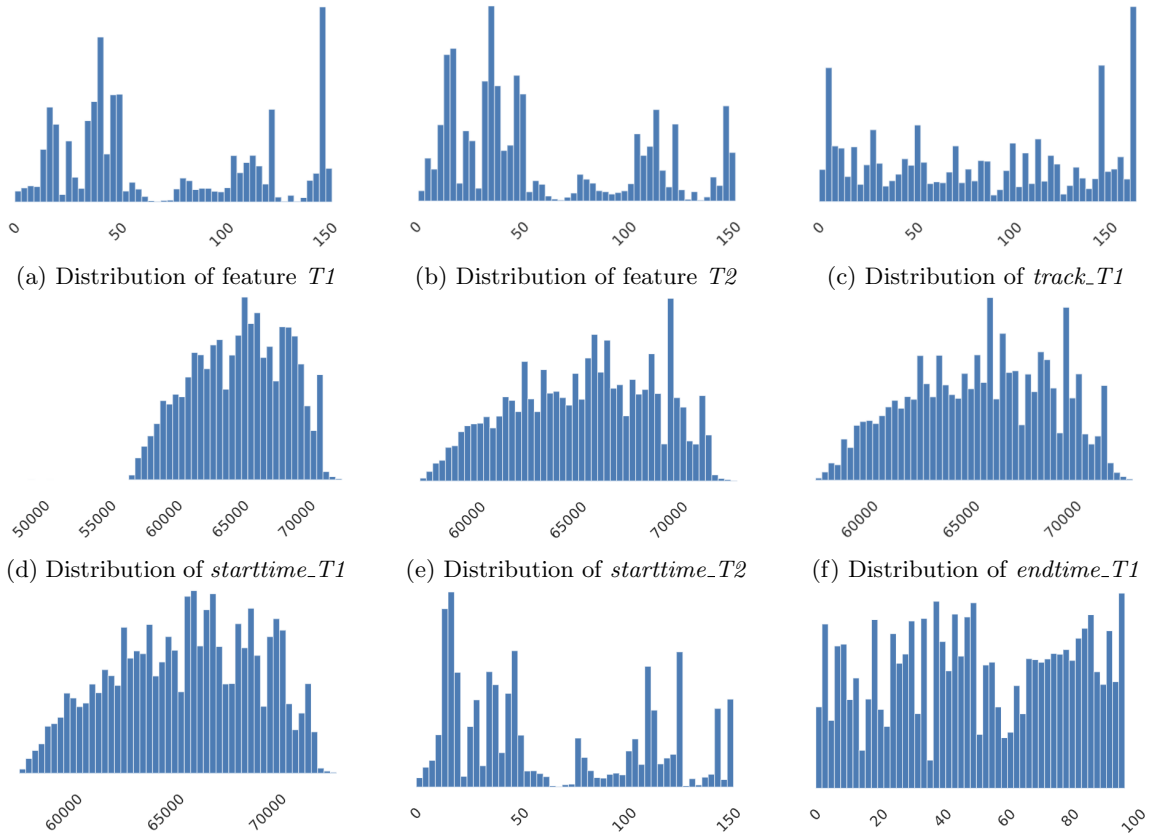
they were created. A dotted line in a *feature lineage graph* indicates the variable that was used to group the data. Table 3 presents some of the synthesized features and their description. A feature may be specific to, e.g., the railway disturbance or the potential conflict. A feature may also be computed over an entire data table, e.g., over the disturbance dataset or the conflict dataset. The features are grouped accordingly in the table.

7 Exploratory data analysis

The features and the corresponding values obtained from the feature engineering phase are stored in a file called `dfs_training_data.csv`, which is our main dataset. The file consists of 194,863 rows and 63 columns, where each row is an example and each column is a certain feature of the examples. The dataset is split into a training set comprising 75% of the examples and a test set comprising 25% of the examples. Thus, the size of training and test sets are 146,147 and 48,716, respectively.

To use the `scikit-learn` library for machine learning, categorical features stored as strings need to be converted to numerical features. One way to encode categorical features is to use a technique called *one hot encoding*. However, one-hot encoding categorical variables with many categories increases the dimensionality of encodings. Since most of our categorical features have many categories, e.g., T1 and T2 can be any of 150 trains, we do not use `OneHotEncoder()` on our categorical features. Instead, we use *label encoding* and ensure that the encoding of train names, sections and tracks across the columns is consistent. For example, the encoding of trains across the columns *T1*, *T2*, and *Disturbances.trainname* is consistent. The code used to encode the categorical columns is listed in Appendix 11.1.

Figure 8 shows the distributions of nine important features in the training dataset, *X_train*. From the distributions of T1 and T2 in Figures 8a and 8b, we can see that not all trains appear as T1 and T2 with equal frequency in our training dataset. Figure 9 shows the distributions of the above nine features in the test dataset, *X_test*, and compares them against those in the training dataset. Figure 10 shows the distribution of the prioritized train across *y_train* and *y_test*. Table 4 shows the top ten trains that are most frequently prioritized in our training and test datasets. Table 5 shows the top 40 triplets $(T1,\ T2,\ y)$ of trains that most frequently appear in the training and test datasets, where y is the prioritized train for a conflict involving T1 and T2.



(a) Distribution of feature *T1*

(b) Distribution of feature *T2*

(c) Distribution of *track_T1*

(d) Distribution of *starttime_T1*

(e) Distribution of *starttime_T2*

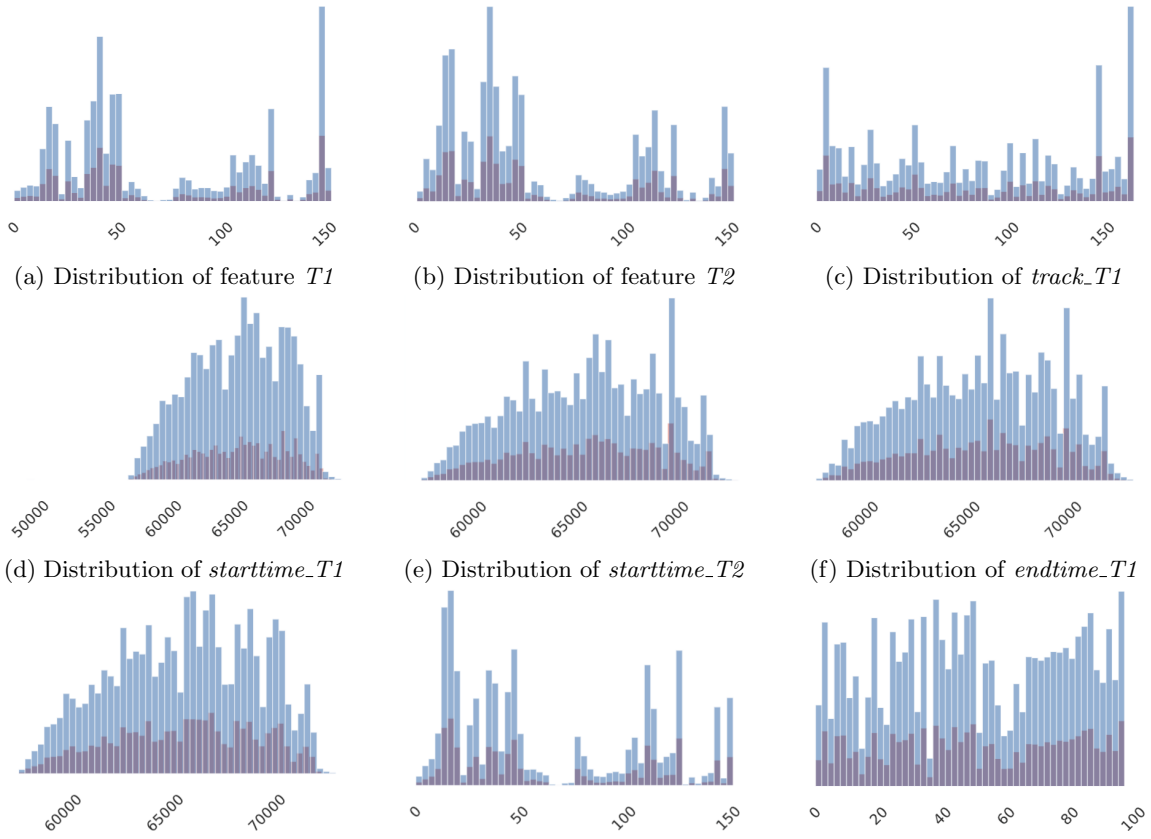
(f) Distribution of *endtime_T1*

(g) Distribution of *endtime_T2*

(h) Distribution of *Disturbances.trainname*

(i) Distribution of *Disturbances.section*

Figure 8: Histograms showing distribution of different features in the training dataset *X_train* containing 146,147 rows.



(a) Distribution of feature *T1*

(b) Distribution of feature *T2*

(c) Distribution of *track_T1*

(d) Distribution of *starttime_T1*

(e) Distribution of *starttime_T2*

(f) Distribution of *endtime_T1*

(g) Distribution of *endtime_T2*

(h) Distribution of *Disturbances.trainname*

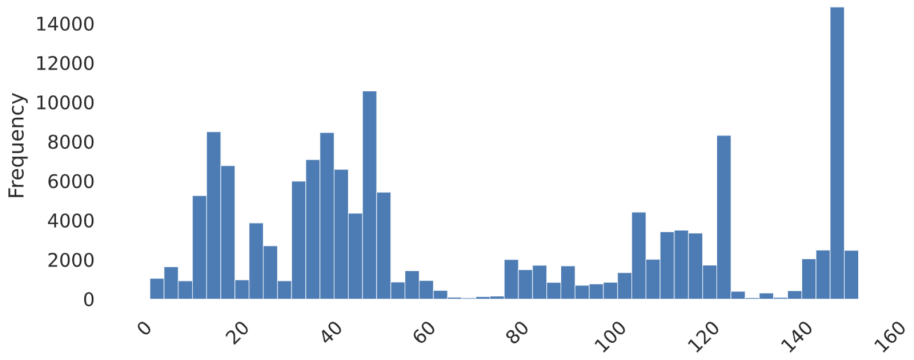
(i) Distribution of *Disturbances.section*

Figure 9: Histograms showing distribution of different features in the test dataset *X_test* containing 48,716 rows, and comparing against the distributions in *X_train*.

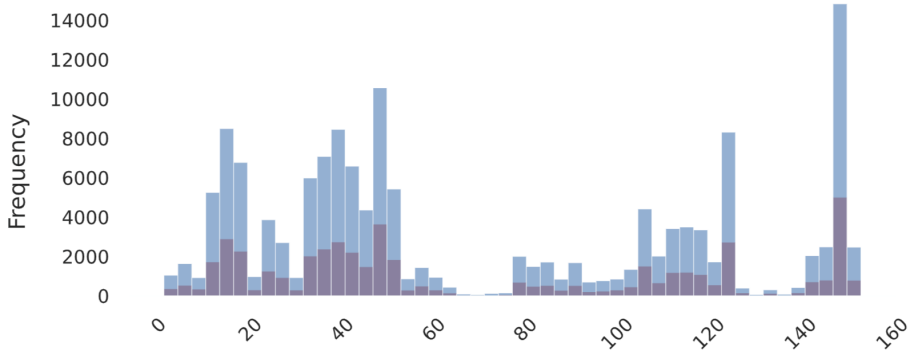
Table 4: Frequency of various trains in *y_train* and *y_test*

y_train target	frequency (%)
149	14,509 (9.9%)
16	8,052 (5.5%)
40	7,955 (5.4%)
37	6,350 (4.3%)
19	6,066 (4.2%)
34	5,307 (3.6%)
13	4,425 (3.0%)
125	4,418 (3.0%)
48	4,263 (2.9%)
50	4,125 (2.8%)
Others	80,677 (55.2%)
All	146,147 (100%)

y_test target	frequency (%)
149	4,895 (10.0%)
16	2,744 (5.6%)
40	2,549 (5.2%)
37	2,150 (4.4%)
19	2,017 (4.1%)
34	1,813 (3.7%)
13	1,470 (3.0%)
125	1,438 (3.0%)
48	1,473 (3.0%)
50	1,438 (3.0%)
Others	26,729 (54.9%)
All	48,716 (100%)



(a) Histogram (50 bins) of the target variable in the training dataset (*y_train*)



(b) Histogram (50 bins) of the target variable in the test dataset (*y_test*) compared against *y_train*.

Figure 10: Frequency distribution of the prioritized train across the dataset

Table 5: Most frequent (T1, T2, y) and their frequency in the training and test datasets

(a) Most frequent (T1, T2, y) in the training data

T1	T2	y	#freq	T1	T2	y	#freq
149	50	149	3532	16	34	16	993
149	37	149	2624	16	149	16	988
149	114	149	2483	114	16	114	949
40	25	40	2226	149	19	149	836
48	34	48	1992	16	37	16	773
41	19	41	1958	152	37	152	763
37	19	37	1708	40	49	40	745
25	40	25	1648	40	13	40	744
42	19	42	1605	45	34	45	737
19	37	19	1533	108	47	108	710
49	40	49	1461	113	149	113	695
50	37	50	1440	37	13	37	694
40	16	40	1392	107	45	107	691
41	16	41	1326	34	113	34	682
40	116	40	1176	19	42	19	671
34	16	34	1102	123	37	123	648
19	149	19	1085	124	34	124	644
125	48	125	1071	37	47	37	611
50	19	50	1051	16	41	16	600
47	37	47	1027	84	37	84	585

(b) Most frequent (T1, T2, y) in the test data

T1	T2	y	#freq	T1	T2	y	#freq
149	50	149	1167	50	19	50	348
149	37	149	903	114	16	114	332
149	114	149	860	16	34	16	303
40	25	40	709	45	34	45	272
48	34	48	670	149	19	149	269
41	19	41	590	16	37	16	261
37	19	37	571	40	13	40	258
42	19	42	545	152	37	152	258
25	40	25	512	113	149	113	248
19	37	19	494	34	113	34	239
50	37	50	493	107	45	107	237
41	16	41	483	108	47	108	233
49	40	49	479	19	42	19	231
40	16	40	428	37	13	37	230
40	116	40	401	123	37	123	228
34	16	34	382	40	49	40	226
47	37	47	376	124	34	124	220
19	149	19	363	37	47	37	217
16	149	16	356	13	40	13	208
125	48	125	351	16	41	16	203

Many preliminary experiments have been conducted to investigate ML models that are a good fit for our data. The preliminary experiments included hyperparameter optimization and cross-validation as well. Some good classification models were obtained from the following learning algorithms: decision trees, random forest, and extra trees. Of these three, the decision trees classifier is well known for being the most interpretable despite being less accurate. Before we discuss some decision tree models for our data, we estimate the importance of the features in our dataset in the next section.

8 Estimating important features

To estimate the importance of various features comprising the dataset, we build a random forest model on the training dataset using `RandomForestClassifier()` of `scikit-learn 1.3.2`. The hyperparameters of the random forest algorithm used in our study are the default values used in the chosen implementation.

Firstly, we build a random forest model on the *X_train* dataset comprising 63 features, which we will refer to as the *rf63feat* model henceforth. The prediction accuracy of *rf63feat* model is 94.9%, i.e., it gave the expected prioritization suggestions for about 46,236 of the 48,716 conflicts in the *X_test* dataset. We use the *rf63feat* model with `SelectFromModel()` from `scikit-learn 1.3.2` to select a subset of the 63 features based on importance weights. `SelectFromModel()` selected

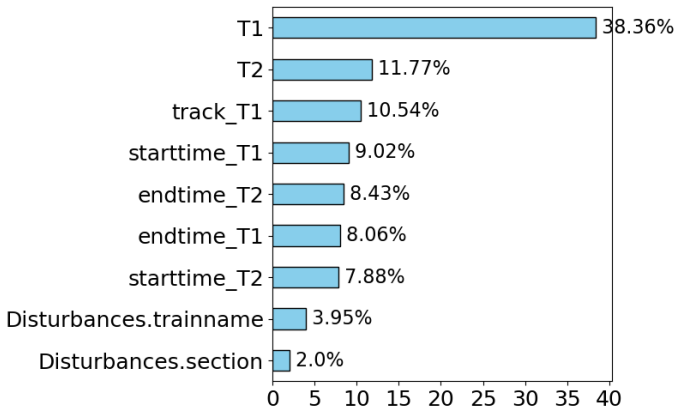


Figure 11: Importance of various features in the *rf9feat* model built with different subsets of features.

the following nine of the 63 features as the most important: (i) *T1* (ii) *T2*, (iii) *track_T1*, (iv) *starttime_T1*, (v) *endtime_T2*, (vi) *endtime_T1*, (vii) *starttime_T2*, (viii) *Disturbances.trainname*, and (ix) *Disturbances.section*.

We build a new random forest model on the *X_train* dataset with the selected nine features, which we will refer to as the *rf9feat* model henceforth. The prediction accuracy of *rf9feat* model is 96.5%, which is a considerable improvement over the *rf63feat* model. Figure 11 summarizes the importance of various features in the *rf9feat* model built with nine features. The feature importance has been computed using the attribute *feature_importances_* of the *rf9feat* model. In the 146,147 conflicts comprising the training dataset, the train that is expected to reach the conflict section first (T1) is prioritized for 82.77% of the conflicts, while the other train T2 is prioritized the remaining 17.23% of the times. This explains why T1 came out to the most important feature and also, more important than T2. Note that in our conflict resolution problem, during a conflict between two trains T1 and T2, we consider prioritizing either T1 or T2 only.

To see the effect of different features on the model's performance, we omit each of the important feature of *rf9feat* model one by one. Tables 6 and 7 show the prediction accuracies of various random forest models that we built with different subsets of features. Based on the above experiment with various subsets of features, we decide to use the following six features of the *rf6feat* model henceforth to create ML models in our study. While resolving conflicts to create a rescheduled timetable, the following six features are indeed crucial to determine whether `T1` or `T2` needs to be prioritized to resolve a conflict.

1. `T1`
2. `T2`
3. `track_T1`
4. `starttime_T1`
5. `endtime_T2`
6. `Disturbances.trainname`

Table 6: Prediction accuracy of various random forest models built with reducing subsets of features

Model	Features used	Accuracy
rf63feat	All 63 features	94.91%
rf9feat	`T1`, `T2`, `track_T1`, `starttime_T1`, `endtime_T2`, `endtime_T1`, `starttime_T2`, `Disturbances.trainname`, `Disturbances.section`	96.5%
rf8feat	All features of *rf9feat* model except `T1`	94%
rf7feat	All features of *rf8feat* model except `track_T1`	93%
rf6feat	All features of *rf7feat* model except `T2`	91%
rf5feat	All features of *rf6feat* model except `starttime_T1`	89%
rf4feat	All features of *rf5feat* model except `endtime_T2`	86%
rf3feat_bad	`starttime_T2`, `Disturbances.trainname`, `Disturbances.section`	58%

Table 7: Prediction accuracy of various random forest models built with different subsets of features

Model	Features used	Accuracy
rf3feat_ok	`T1`, `Disturbances.trainname`, `Disturbances.section`	80%
rf1feat	`T1`	83%
rf2feat	`T1`, `T2`	86%
rf3feat	`T1`, `T2`, `track_T1`	93%
rf4feat	`T1`, `T2`, `track_T1`, `starttime_T1`	95.93%
rf5feat	`T1`, `T2`, `track_T1`, `starttime_T1`, `Disturbances.trainname`	96.44%
*rf5feat_*	`T1`, `T2`, `track_T1`, `starttime_T1`, `endtime_T2`	96.47%
rf6feat	`T1`, `T2`, `track_T1`, `starttime_T1`, `endtime_T2`, `Disturbances.trainname`	96.77%
rf7feat	All six features of the *rf6feat* model and `Disturbances.section`	96.52%

In the 48,716 conflicts comprising the test dataset, the train that is expected to reach the conflict section first (T1) is prioritized for 83.13% of the conflicts, while T2 is prioritized 16.87% of the times. If we create a hypothetical model that always suggests the decision maker to prioritize T1 for every conflict, that model will have an accuracy of 83.13% on the *X_test* dataset. This explains how the *rf1feat* model that considers T1 as the only feature has an accuracy of 83%. We will refer to the above hypothetical model as the *fcfs1feat* model henceforth where "fcfs" means "first come first served".

9 Classification models

9.1 Decision tree models

Decision trees are machine learning models that are well-known for their interpretability, given that they are not large. Hence, we investigate decision tree models to suggest a decision maker which train to prioritize during conflict resolution.

Firstly, we create decision tree models using only the most important feature, i.e., `T1`. Upon fitting the `DecisionTreeClassifier()` with default (hyper)parameters on the *X_train* dataset, we

obtain a decision tree model with accuracy of 83.33%, which we will refer to as the *dt1feat* model henceforth. Comparing the *dt1feat* model with the *fcfs1feat* model shows that the accuracy of the *dt1feat* model is only marginally higher than the other model while the interpretability is low. The reason for the low interpretability is because the *dt1feat* model is a large decision tree with a depth of 31 and 301 nodes, which may be difficult to interpret. When we limit the maximum depth of the *dt1feat* model using `DecisionTreeClassifier(max_depth=4)` for a smaller and a more interpretable decision tree, the model's accuracy falls down to 34.53%. This is not surprising given that we are using only one feature: T1. Table 8 shows the accuracy of various decision tree models created using T1 as the feature and Figure 12 shows two of those decision trees with prediction accuracies of 14.66% and 19.87% on the *X_test* dataset. The accuracy and the actual decision trees in Figure 12 are not meaningful for train prioritization in the context of conflict resolution and rescheduling. One of the reasons is that for a conflict between trains T1 and T2, we only prioritize either T1 or T2. The two decision trees in Figure 12 fail to capture this aspect of our problem. This is not surprising and is expected of decision trees created using only one feature and depth not more than two.

The reason why the learning algorithm creates decision trees as in Figure 12 can be understood from Table 5b that shows the most frequently prioritized trains in the test dataset. A portion of this table is duplicated next to Figure 12. Since Train 149 is prioritized at least 1167 + 903 + 860 = 2930 times and Train 40 is prioritized at least 709 times, the decision tree has those trains in the leaf nodes to maximize the overall prediction accuracy of the model over *X_test*.

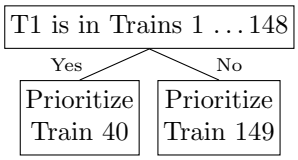
Next, we create a decision tree model with `DecisionTreeClassifier()` using the six features of the *rf6feat* model mentioned in Section 8. The resulting model, trained on the *X_train* dataset has an accuracy of 96.27%. This decision tree has a depth of 35 and 17,645 nodes. When we limit the maximum depth of the *dt6feat* model for a smaller and a more interpretable decision tree, the model's accuracy falls down as shown in Table 9.

Table 8: Decision tree models with various depths created using T1 as the feature

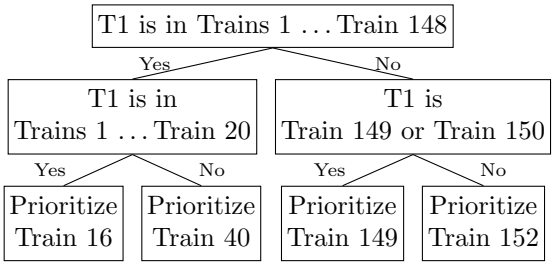
dtree1feat model created using	Depth	Nodes	Accuracy
`DecisionTreeClassifier()`	31	301	83.33%
`DecisionTreeClassifier(max_depth=4)`	4	23	34.53%
`DecisionTreeClassifier(max_depth=3)`	3	13	25.74%
`DecisionTreeClassifier(max_depth=2)`	2	7	19.87%
`DecisionTreeClassifier(max_depth=1)`	1	3	14.66%

Table 9: Decision tree models with various depths created using the six features

dtree6feat model created using	Depth	Nodes	Accuracy
`DecisionTreeClassifier()`	35	17,645	96.27%
`DecisionTreeClassifier(max_depth=18)`	18	10,087	91.34%
`DecisionTreeClassifier(max_depth=9)`	9	711	66.68%
`DecisionTreeClassifier(max_depth=5)`	5	63	42.22%
`DecisionTreeClassifier(max_depth=3)`	3	15	25.70%
`DecisionTreeClassifier(max_depth=2)`	2	7	19.87%



(a) *dtree1feat* model with max_depth = 1 (accuracy is 14.66%)



(b) *dtree1feat* model with max_depth = 2 (accuracy is 19.87%)

Figure 12: Decision tree models created with T1 as the feature

Most frequent (T1, T2, y) in the test data

T1	T2	y	#freq
149	50	149	1167
149	37	149	903
149	114	149	860
40	25	40	709
48	34	48	670
41	19	41	590
37	19	37	571
42	19	42	545
...	...	...	...

The *dtree6feat* model created with a `max_depth = 2` was the same as the *dtree1feat* model with `max_depth = 2` shown in Figure 12b. The decision tree with `max_depth = 3` for the *dtree6feat* model is shown in Figure 13. The prediction accuracy of this decision tree is very low at 25.70% and the prioritization decisions made by the tree do not look meaningful or reasonable in the context of conflict resolution and rescheduling. Instead of investigating larger decision trees that may provide higher accuracy at the cost of interpretability, we investigate a more powerful machine learning method in the next section and explain the model's decisions using model-agnostic methods.

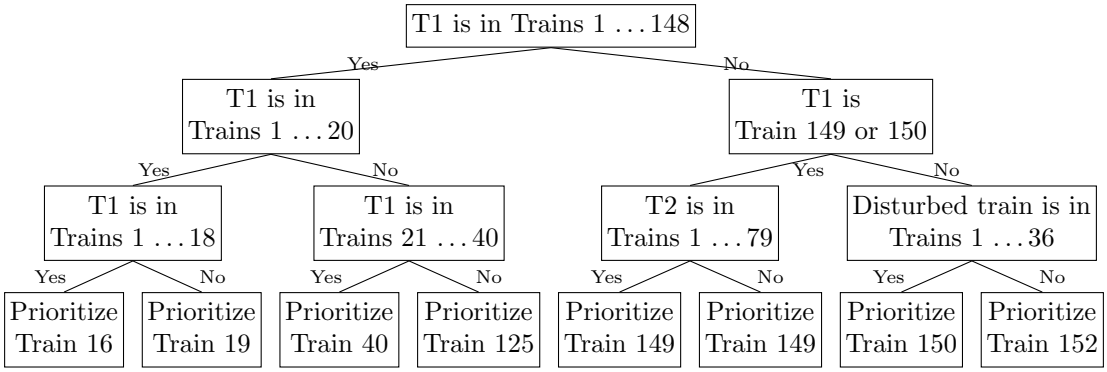


Figure 13: *dtree6feat* model with max_depth = 3 (accuracy is 25.70%)

9.2 Random forest models

Random forest is a machine learning algorithm that constructs multiple decision trees and combines their outputs to arrive at a more accurate prediction. Random forests are well-known for preventing overfitting, which is a common problem with decision tree based models. In our study, we use the `RandomForestClassifier()` of `scikit-learn 1.3.2` with default (hyper)parameters to create our models. An example of a hyperparameter is the number of decision trees in the forest (represented by `n_estimators`), which is 100 by default.

We already built a random forest model using six features while estimating the importance of features in Section 8. This model, that we called the *rf6feat* model gave a prediction accuracy of 96.77%. In this section, we investigate, analyze, and explain the *rf6feat* model's predictions, i.e., suggestions about which train to prioritize. To explain the model's predictions, we use a method called *SHAP* (Shapley additive explanations). SHAP is a widely employed method that can explain predictions made by a ML model. In our problem context, SHAP can explain why a potential conflict receives a certain train prioritization suggestion and can quantify the positive or negative contributions of each feature towards that suggestion. In our problem context, this provides transparency by allowing a train dispatcher to understand why the ML model gave a specific prioritization suggestion for the given input. The train dispatcher can also verify from the explanation that the relationship between the feature values and the model's suggestion are consistent from a railway domain knowledge perspective.

The SHAP values are computed using `shap 0.42.1` Python package as follows. The computed `shapvalues` is a list of arrays (`numpy.ndarray`), where each array corresponds to a target class and has the same shape as our dataframe.

```
my_explainer = shap.TreeExplainer(model=rf6feat)
shapvalues = my_explainer.shap_values(my_dataframe)
```

In the remainder of this section, we discuss two specific disturbance scenarios out of the 18,000 scenarios that were used to create our conflict dataset. We will discuss the potential conflicts generated by the disturbances and the model's suggestions for each potential conflict.

9.2.1 Disturbance scenario 9958

In disturbance scenario 9958, Train 1279 incurs a delay of 15 min at the line section E-DAT connecting Eslov (E) and Damstorp (DAT). Table 10 shows the earliest potential conflict in the train timetable created by the disturbance. The reason for the potential conflict is that Trains 1279 and 1083 are to occupy track#2 of section E-DAT during overlapping time intervals (see Table 10). Trains 1279 and 1083 were encoded in the dataset as Trains 57 and 26, respectively. The section E-DAT was encoded as section 28, and the second track of E-DAT where there is a potential conflict was encoded as track 42. In the remainder of the section, we will refer to the trains and sections with their encoded numbers and original names interchangeably, depending on the context.

When the conflict in Table 10 is given as input to the *rf6feat* model, it suggests to prioritize Train 57 with a *class probability* of 100%. To understand why the model made the particular prediction, i.e., suggested the decision to prioritize Train 57 for the specific conflict, we present the corresponding SHAP waterfall plot in Figure 14.

Table 10: Details of the first potential conflict created by disturbance scenario 9958

T1	T2	Disturbance		T1			T2	
		trainname	section	track	starttime	endtime	starttime	endtime
1279 (57)	1083 (26)	1279 (57)	E-DAT (28)	E-DAT#2 (42)	59760	60489	60360	60488

We start at the bottom of the waterfall plot in Figure 14. The plot starts with the base value of 0.003, called $E[f(X)]$. This base value of 0.003 can be interpreted as follows by a train dispatcher. If the model knows nothing about the feature values, then it would suggest to prioritize Train 57 with a confidence of 0.3%. The base value for a suggestion, e.g., *"Prioritize Train 57"*, is computed based on the the training dataset.

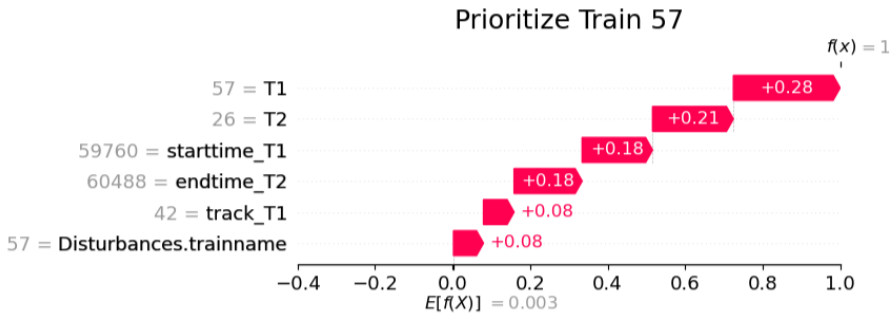


Figure 14: Scenario 9958: A waterfall plot to understand why the model suggests to prioritize Train 57 when resolving conflict 1 between Trains 57 and 26 at E-DAT#2

Each row from the bottom of a waterfall plot shows how the contribution of each feature moves the base value to the model's output. It can be interpreted as the model's suggestion to prioritize Train 57 for the given conflict being either strengthened or weakened. For the conflict in Table 10, the disturbed train being Train 57 contributes 8% to the model's suggestion to *prioritize Train 57* (see Figure 14), thus increasing the probability to around 8%. The conflict track being track 42 also contributes 8% to the model's suggestion (see Figure 14), thus increasing the probability to around 16%. The times when T2 is scheduled to exit the conflict section and T1 is scheduled to enter the conflict section, both contribute 18% separately to the model's suggestion, thus increasing the suggestion's probability to around 52%. T2 being Train 26 contributes 21% to the model's suggestion, thus increasing the probability to around 73%. Finally, T1 being Train 57 increases the probability to 100% (when rounding errors are considered). Thus, $f(x)$ has a value of 1 in Figure 14.

The value of T1 = 57 has the largest effect on the model's suggestion to prioritize Train 57. The training dataset comprises 364 conflicts from various disturbance scenarios where Train 57 is T1, i.e., the train that is scheduled to occupy the conflict section first. 79.4% of these conflicts (289 conflicts) were resolved by prioritizing Train 57. Thus, it is understandable why the probability of prioritizing Train 57 is increased the most due to T1 being 57. The value of T2 = 26 also makes a significant contribution to the model's suggestion. The training dataset has 131 conflicts

between Trains 57 and 26 where T1 = 57 and T2 = 26. When resolving these conflicts, Train 57 was prioritized almost 90% of the times.

When it comes to the alternative decision, i.e., prioritizing Train 26 to resolve the conflict in Table 10, the *rf6feat* model computes a probability of 0% for that decision. The waterfall plot in Figure 15 shows the contribution of each feature value towards this probability. The plot starts with a base value of 0.003, which means a probability of 0.3%. T1, i.e, the first train scheduled to enter the conflict section among the two trains in conflict, being Train 57 reduced this probability by 1% (see -0.01 in Figure 15). The time when T1 is scheduled to enter the conflict section further reduces the probability by 2% (see -0.02 in Figure 15). T2 being Train 26 contributes 2% to the model's suggestion, thus bringing the total probability to 0 (considering the rounding errors).

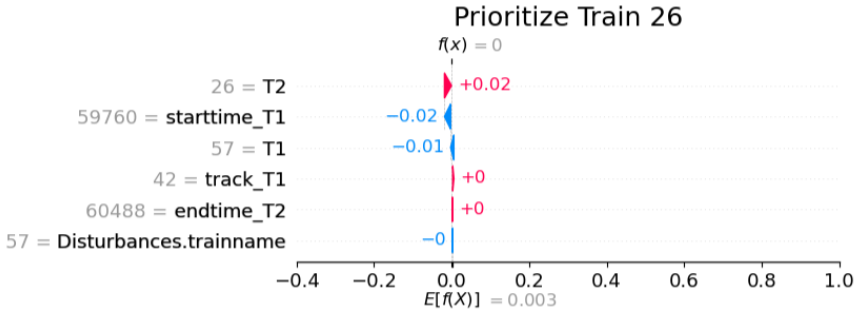


Figure 15: Scenario 9958: A waterfall plot to understand why the model does not suggest to prioritize Train 26 when resolving conflict 1 between Trains 57 and 26 at E-DAT#2

A rescheduling algorithm (Josyula, 2021) is made to update the train timetable after resolving the conflict by prioritizing Train 57 as per the *rf6feat* model's suggestion. The algorithm prioritizes Train 57 by making Train 26 wait for Train 57 at the double-track section E-DAT. The algorithm updates the timetable accordingly and detects the earliest potential conflict again in the updated timetable. Table 11 shows the details of this conflict, which is at the adjacent station DAT. The reason for the potential conflict is that Trains 26 and 57 are both scheduled to occupy track#2 at station DAT during overlapping time intervals (see Table 11).

Table 11: Details of the second potential conflict created by disturbance scenario 9958

T1	T2	Disturbance		T1			T2	
		trainname	section	track	starttime	endtime	starttime	endtime
1083 (26)	1279 (57)	1279 (57)	E-DAT (28)	DAT#2 (35)	60489	60489	60489	60489

When the above conflict is given as input to the *rf6feat* model, it suggests to prioritize Train 57 with a *class probability* of 94%. To understand why the model made the particular prediction, we present the corresponding SHAP waterfall plot in Figure 16. The plot can be interpreted in the same way as before.

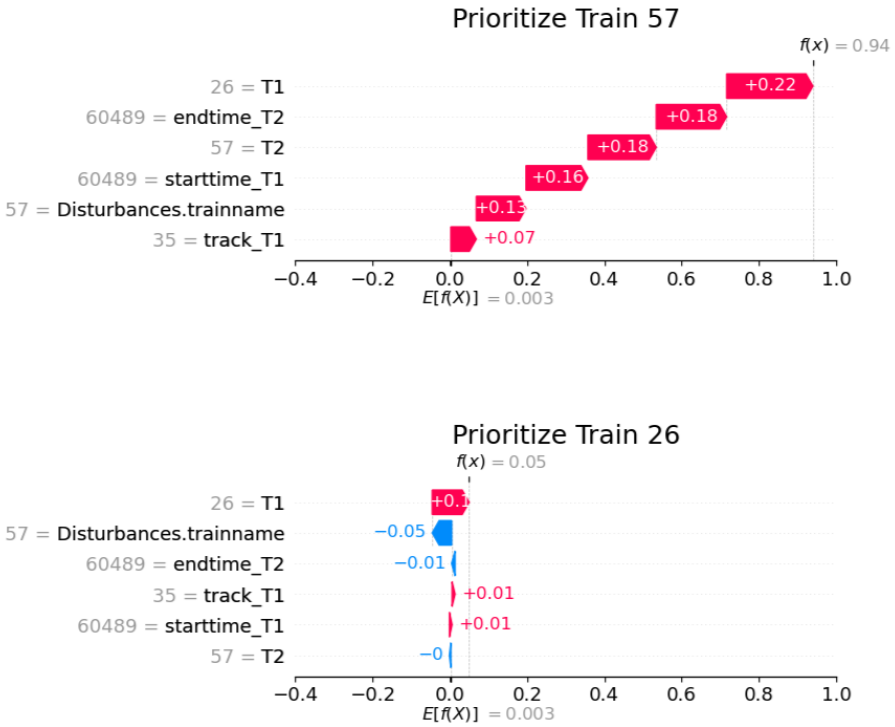


Figure 16: Scenario 9958: A waterfall plot to understand why the model suggests to prioritize Train 57 when resolving conflict 2 between Trains 57 and 26 at DAT#2

Notice from the plot that the probabilities to prioritize Train 57 and Train 26 are $f(x) = 0.94$ and 0.05, respectively. Inspecting the output of the *rf6feat* model shows that the model assigns the remaining probability of 1 - 0.94 - 0.05 = 0.01 to prioritize Train 28. This is not meaningful to us because in our problem context the train to be prioritized can only be either T1 or T2. This could have been avoided by treating our problem as multiple binary classification problems instead of a single multi-class classification problem.

The rescheduling algorithm of Josyula (2021) is used to update the train timetable again after resolving the conflict by prioritizing Train 57 as per the *rf6feat* model's suggestion. This time, the rescheduling algorithm prioritizes Train 57 by reallocating the track of Train 26 at the station DAT as there is an empty track available at the station. The rescheduling algorithm updates the timetable accordingly and checks for potential conflicts. There are no other conflicts and so the last updated timetable is a conflict-free rescheduled timetable, shown in Figure 17. The rescheduled timetable shows how Train 26 (i.e., Train 1083 in Figure 17) is made to wait at E-DAT. The train's path is shown in bold black dotted line in Figure 17.

Both the conflicts given as input to the *rf6feat* model so far were in the training dataset, and the model made correct predictions for these two conflicts. In the next section, we will use the *rf6feat* model on conflicts from the test dataset that it has not seen before.

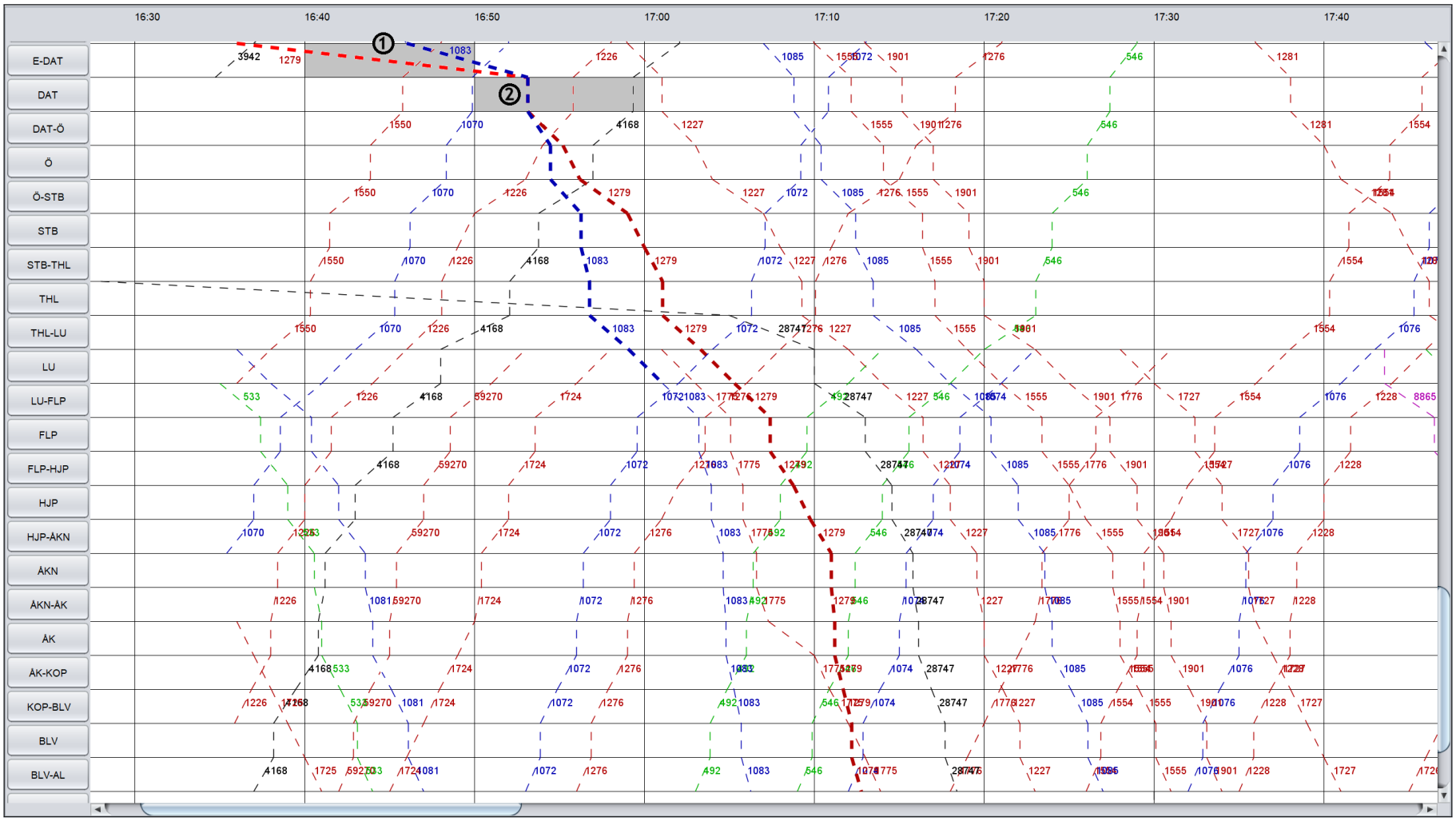


Figure 17: A portion of the rescheduled timetable for Scenario 9958. Total final delay in the rescheduled timetable is 8 min 19 sec. The dotted lines are the train paths after rescheduling. Significant delays as a result of rescheduling are shown in bold dotted lines.

9.2.2 Disturbance scenario 345

In disturbance scenario 345, Train 533 incurs a delay of 25 min at the line section MLB-SÖLA connecting Mellby (MLB) and Sösdala (SÖLA). Table 12 shows the earliest potential conflict in the train timetable created by the disturbance. The reason for the potential conflict is that Trains 533 and 1083 are to occupy track#2 of section MLB-SÖLA during overlapping time intervals (see Table 12). Trains 533 and 1083 were encoded in the dataset as Trains 5 and 26, respectively. The section MLB-SÖLA was encoded as section 55, and the second track of MLB-SÖLA with the potential conflict was encoded as track 98. In the remainder of the section, we will refer to the trains and sections with their encoded numbers and original names interchangeably, depending on the context.

Table 12: Details of the first potential conflict created by disturbance scenario 345

T1	T2	Disturbance		T1			T2	
		trainname	section	track	starttime	endtime	starttime	endtime
533 (5)	1083 (26)	533 (5)	MLB-SÖLA (55)	MLB-SÖLA#2 (98)	58294	59599	59173	59289

The conflict in Table 12 belongs to the test dataset and the *rf6feat* model has not seen it during the training phase. When the above conflict is given as input to the model, it suggests to prioritize Train 5 with a *class probability* of 100%. To understand why the model made the particular prediction for the specific conflict, we present the corresponding SHAP waterfall plot in Figure 18.

The rescheduling algorithm of (Josyula, 2021) is used to resolve the conflict by prioritizing Train 5 as per the *rf6feat* model's suggestion and to update the train timetable. The rescheduling algorithm prioritizes Train 5 by making Train 26 wait for Train 5 at the double-track section MLB-SÖLA. This update can be seen in the final rescheduled timetable shown in Figure 22.

After updating the timetable, the rescheduling algorithm again detects the earliest potential conflict. Table 13 shows the details of this conflict, which is at the Sösdala station (SÖLA). The reason for the potential conflict is that Trains 26 and 5 are both scheduled to occupy track#2 at station SÖLA during overlapping time intervals (see Table 13).

Table 13: Details of the second potential conflict created by disturbance scenario 345

T1	T2	Disturbance		T1			T2	
		trainname	section	track	starttime	endtime	starttime	endtime
1083 (26)	533 (5)	533 (5)	MLB-SÖLA (55)	SÖLA#2 (124)	59599	59599	59599	59599

The conflict in Table 13 belongs to the training dataset and the *rf6feat* model has already seen it during the training phase. When this conflict is given as input to the model, it suggests to prioritize Train 5 with a class probability of 99%. To understand why the model made the particular prediction for the specific conflict, see the corresponding SHAP waterfall plot in Figure 19.

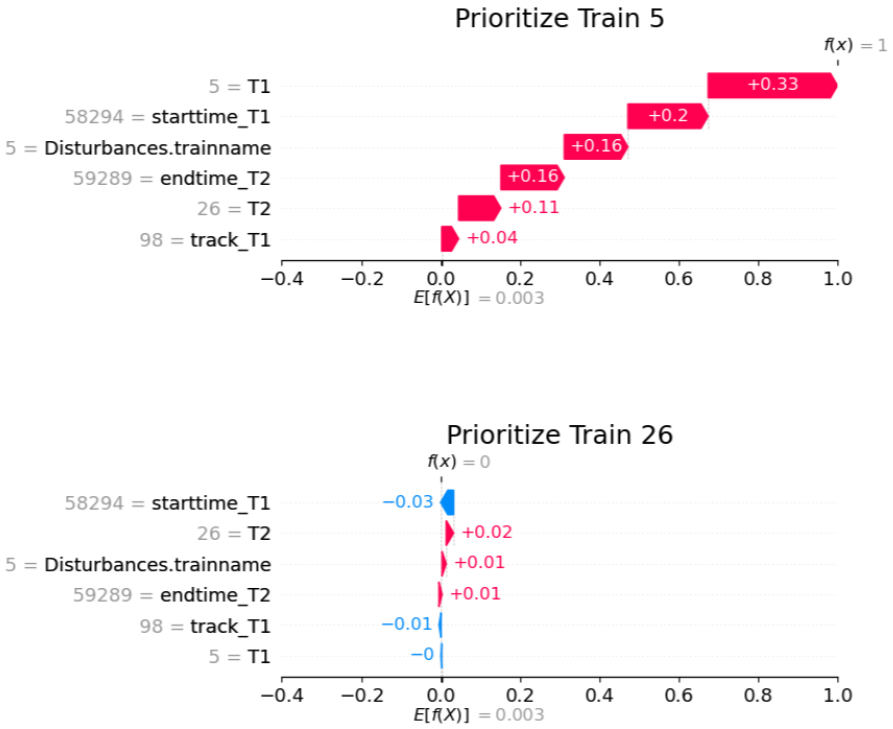


Figure 18: Scenario 345: A waterfall plot that explains why the model suggests prioritizing Train 5 to resolve potential conflict 1 at track 2 of section MLB-SÖLA

The aforementioned rescheduling algorithm is used to resolve the conflict by prioritizing Train 5 as per the *rf6feat* model's suggestion and to update the train timetable. The algorithm prioritizes Train 5 and reschedules Train 26 by reassigning a different track to it at the Sösdala station. The algorithm performs a track reassignment to Train 26 instead of retiming it because there is an empty track available at Sösdala during the start and end times of Train 26 at that station. Due to this track reassignment to Train 26 at Sösdala, the train paths of Trains 5 and 26 (i.e., Trains 533 and 1083, respectively) seem to be overlapping in the rescheduled timetable (see section SÖLA in Figure 22).

After updating the train timetable, the rescheduling algorithm detects the next earliest potential conflict, the details of which are shown in Table 14. This conflict is at the line section SÖLA-VÄD, again between Trains 26 and 5.

The conflict in Table 14 belongs to the test dataset and the *rf6feat* model has not seen it during the training phase. When this conflict is given as input to the model, it suggests to prioritize Train 5 with a class probability of 98%. To understand why the model made the particular prediction for the specific conflict, see the corresponding SHAP waterfall plot in Figure 20. Our rescheduling algorithm is used to resolve the conflict by prioritizing Train 5 as per the *rf6feat* model's suggestion and to update the train timetable. The algorithm prioritizes Train 5 and reschedules Train 26 by reassigning a different track to it at SÖLA-VÄD section. The algorithm performs a track reassignment to Train 26 instead of retiming it because there is an empty track available at the

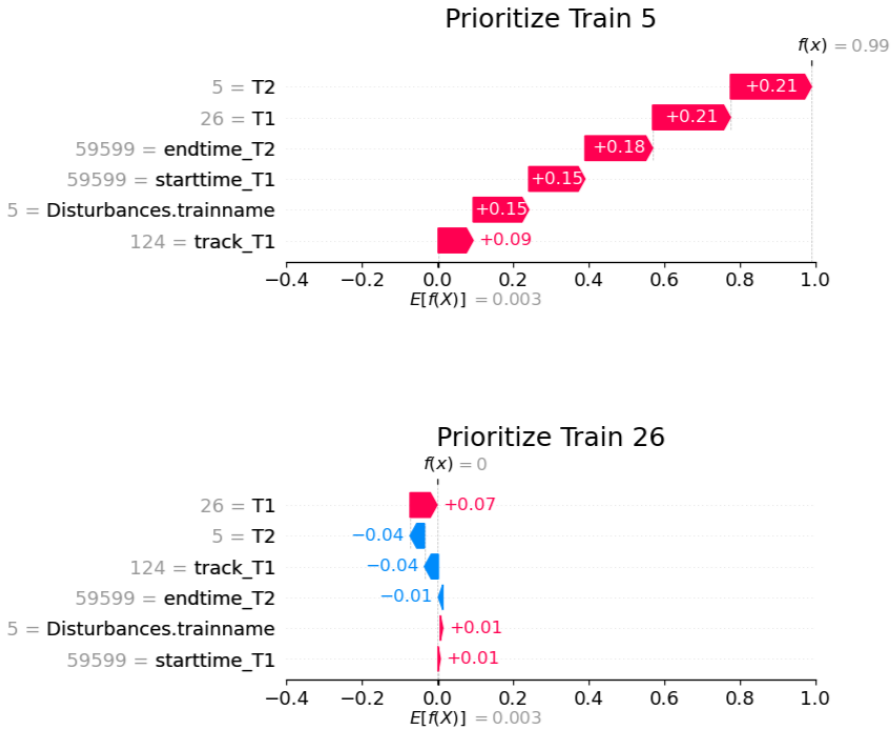


Figure 19: Scenario 345: A waterfall plot that explains why the model suggests prioritizing Train 5 to resolve potential conflict 2 at track 2 of station SÖLA

section SÖLA-VÄD. See the relevant portion of the rescheduled timetable at section SÖLA-VÄD in Figure 22.

Table 14: Details of the third potential conflict created by disturbance scenario 345

T1	T2	Disturbance		T1			T2	
		trainname	section	track	starttime	endtime	starttime	endtime
1083 (26)	533 (5)	533 (5)	MLB-SÖLA (55)	SÖLA-VÄD#2 (126)	59599	59651	59599	59645

Finally, after updating the train timetable, the rescheduling algorithm detects the next earliest potential conflict, the details of which are shown in Table 15. This conflict is at the line section ÅKN-ÅK, between Trains 57 and 5. The reason for the potential conflict is due to a small overlap in the time intervals of Trains 1279 and 533 at track#2 of section ÅKN-ÅK (see Table 15).

The conflict in Table 15 belongs to the test dataset and the *rf6feat* model has not seen it during the training phase. When this conflict is given as input to the model, it suggests to prioritize Train 57 with a class probability of 94%. To understand why the model made the particular

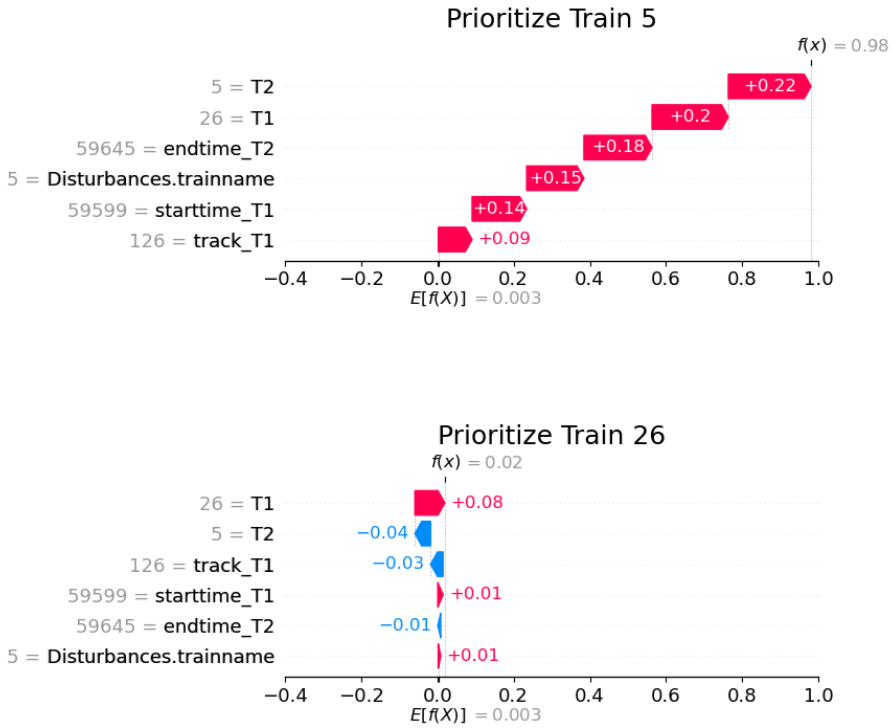


Figure 20: Scenario 345: A waterfall plot that explains why the model suggests prioritizing Train 5 to resolve potential conflict 3 at track 2 of section SÖLA-VÄD

prediction for the specific conflict, see the corresponding SHAP waterfall plot in Figure 21. Our rescheduling algorithm is used to resolve the conflict by prioritizing Train 57 as per the *rf6feat* model's suggestion and to update the train timetable. The algorithm prioritizes Train 57 (i.e., Train 1279) by retiming Train 5 (i.e., Train 533) accordingly. See the relevant portion of the rescheduled timetable at section ÅKN-ÅK in Figure 23.

Table 15: Details of the fourth potential conflict created by disturbance scenario 345

T1	T2	Disturbance		T1			T2	
		trainname	section	track	starttime	endtime	starttime	endtime
1279 (57)	533 (5)	533 (5)	MLB-SÖLA (55)	ÅKN-ÅK#2 (155)	61055	61080	61063	61073

Figure 21: Scenario 345: A waterfall plot that explains why the model suggests prioritizing Train 57 to resolve potential conflict 4 at track 2 of section ÅKN-ÅK

In the above-discussed disturbance scenario 345, the *rf6feat* model gave prioritization suggestions for four conflicts, three of which it had not seen before. The suggestions led to a rescheduled timetable with a total final delay of 16 min 39 sec. The devised ML model showed that it is feasible to obtain a good rescheduled timetable following its data-driven suggestions. However, note that the model had to work along with an existing rescheduling algorithm in this study.

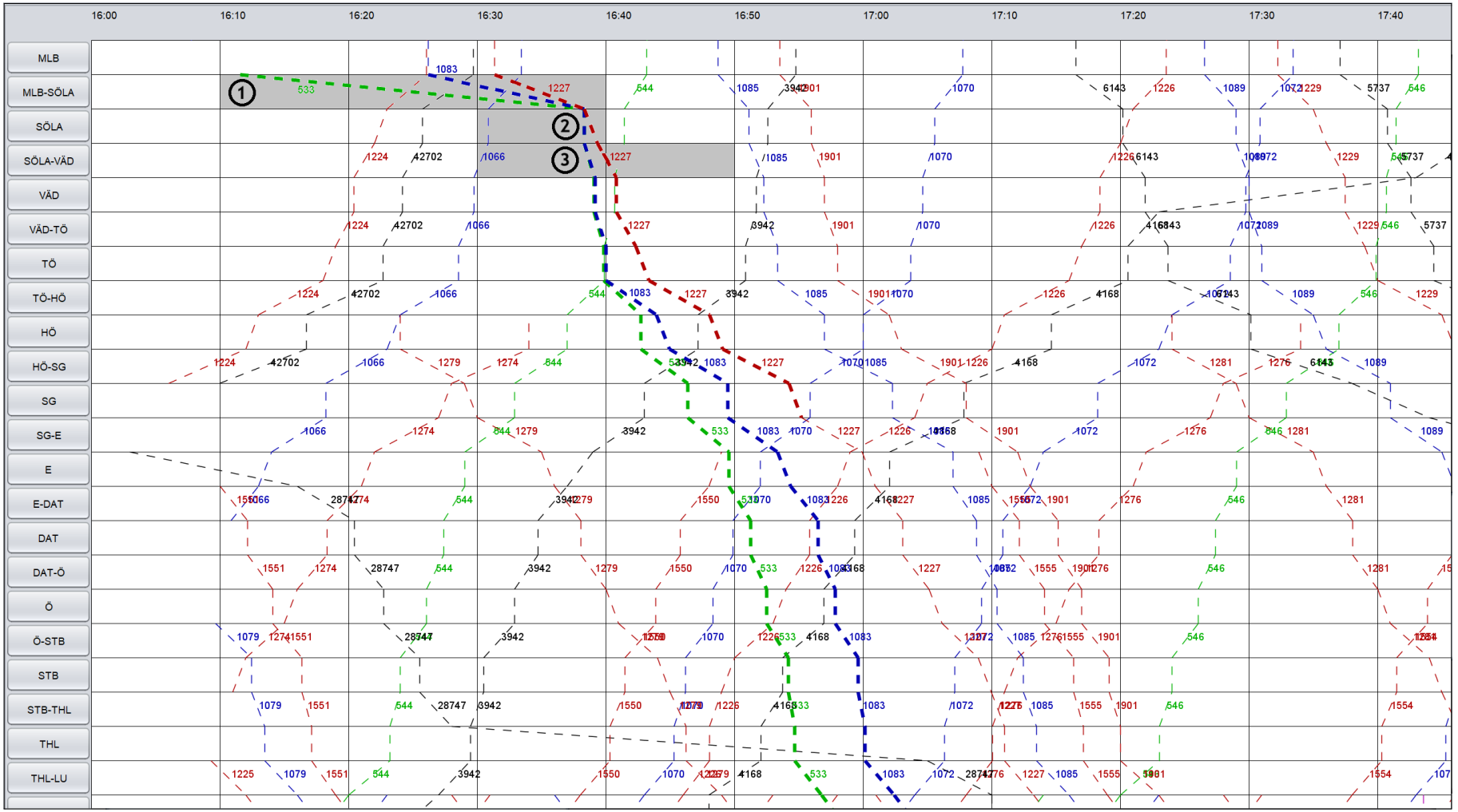


Figure 22: A portion of the rescheduled timetable for Scenario 345 showing the three resolved conflicts. Total final delay in the rescheduled timetable is 16 min 39 sec.

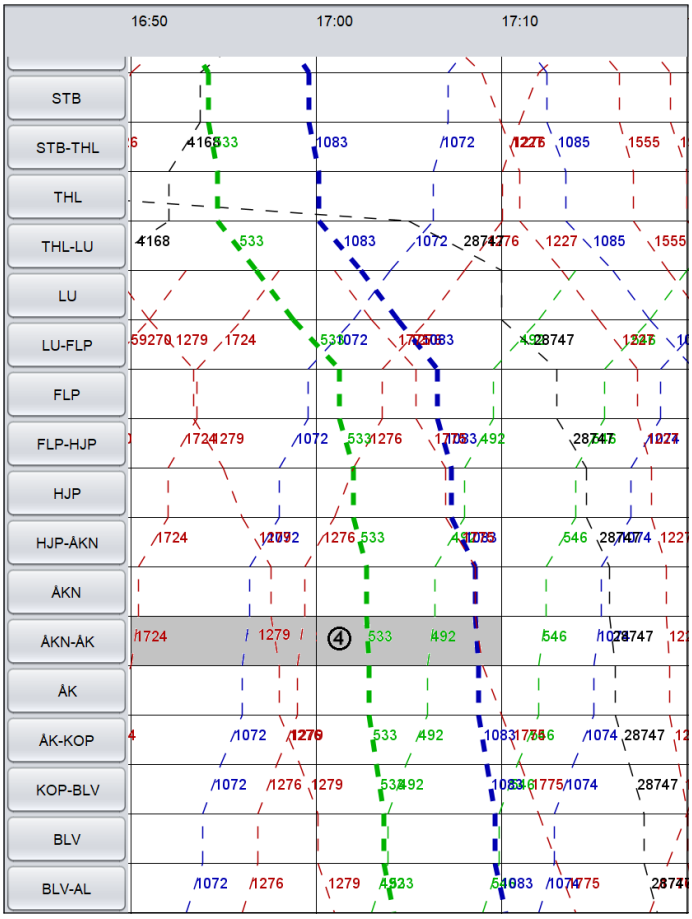


Figure 23: A portion of the rescheduled timetable for Scenario 345 showing the fourth resolved conflict between Trains 1279 and 533 at section ÅKN-ÅK.

10 Conclusions and Future work

This report presents a summary of the MATRIX project (Jan 2023–Jan 2024) in which we conducted an experimental study on machine learning based decision support for train traffic disturbance management. In this research project, we created ML models that can assist train dispatchers in resolving potential conflicts in a train timetable.

We created a large artificial training dataset using around 18,000 disturbance scenarios generated in a 2021 train timetable on the Karlskrona-Hässleholm-Malmö stretch. We devised ML models for data-driven decision making to resolve potential conflicts in a train timetable during rescheduling. When working alongside an existing rescheduling algorithm, the ML models showed capable of resolving potential conflicts during a disturbance scenario and arriving at good rescheduled timetables.

Deep feature synthesis proved to be a good fit for our problem and helped to synthesize many

meaningful features from our relational data. However, our study showed that most of the synthesized features were not necessary for creating decision tree and random forest models with good prediction accuracy. On the other hand, omitting most of the synthesized features increased the predictive power of the resulting ML models.

The experiments carried out in the MATRIX project show that ML algorithms offer many opportunities to the train traffic domain during train timetable rescheduling. In our experiments, using decision tree and random forest algorithms, we created models with a prediction accuracy of more than 96%. Accurate decision tree models for our problem data resulted in fairly large decision trees, e.g., trees with depth 35 and 17,645 nodes, due to which the benefit of interpretability offered by a decision tree model was lost. A random forest model that was created using six selected features gave a prediction accuracy of 96.77%. The model's decisions were then explained using a model-agnostic method called SHAP.

The process of making a prediction by applying a trained ML model to an example is called *inference*. In our context, the process of the ML model predicting the train to be prioritized for a given conflict is called inference. In real-time applications such as an ML-based application for train timetable rescheduling, inference time of the ML model is an important metric. When rescheduling a train timetable during a disturbance scenario, conflict resolution takes place many times, typically iteratively. A higher inference time for the ML model used during conflict resolution may slow down the rescheduling process. In the future, while assessing the suitability of various ML algorithms, inference time is a metric that also needs to be considered.

Our study shows that ML algorithms offer several opportunities for intelligent data-driven solutions for train timetable rescheduling. ML-based approaches for train prioritization during conflict resolution showed to be feasible and effective in producing good rescheduled timetables.

11 Appendix

11.1 Numerical encoding of categorical columns

```
def label_encode(df: pd.DataFrame):
    dfnew = pd.DataFrame(columns=df.columns,
                data=LabelEncoder().fit_transform(df.values.flatten()).reshape(df.shape))
    return dfnew

def make_data_numerical(df: pd.DataFrame):
    trackname_cols = ['track_T1', 'Infra.MODE(conflicts.track_T1)',
                      'Infra.MODE(Disturbances.conflicts.track_T1)',
                      'Infra.MODE(Timetable.track)']
    secname_cols = ['section', 'Disturbances.section',
                    'Infra.MODE(Disturbances.conflicts.section)']
    sectype_cols = ['Infra.sectype', # NWK, LOC
                    'Disturbances.Infra.sectype'] # NWK, LOC
    # Manually append the following categorical columns
    trainname_cols = ['T1', 'T2', 'Disturbances.trainname',
                      'Infra.MODE(conflicts.T1)', 'Infra.MODE(Disturbances.trainname)',
                      'Infra.MODE(Disturbances.conflicts.T1)',
                      'Infra.MODE(Disturbances.conflicts.T2)',
                      'Infra.MODE(Timetable.trainname)', 'target']
```

```
    df_numerical_cols = df.drop(columns=trackname_cols + secname_cols + sectype_cols
    + trainname_cols, axis=1)

    df_numencoded_cols = pd.DataFrame()
    # Concatenate the DataFrames along rows (axis=0)
    for cols in [trainname_cols, secname_cols, trackname_cols, sectype_cols]:
        df_numencoded_cols = pd.concat([df_numencoded_cols, label_encode(df[cols])], axis=1)
    df_new = pd.concat([df_numencoded_cols, df_numerical_cols], axis=1)
    # Bring the target column to the end
    target_column = df_new.pop('target') # Save the column
    # Add the extracted column at the last column of the DataFrame
    df_new['target'] = target_column
    return df_new
```

11.2 Training random forest models

```

import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from tpot.export_utils import set_param_recursive
from sklearn.feature_selection import SelectFromModel
from pathlib import Path
from sklearn.metrics import accuracy_score

def train_model(X_train, y_train, modelname):
    mypipeline = make_pipeline(RandomForestClassifier())
    print(f"Model name is {modelname}_{len(X_train.columns)}")
    # Fix random state for all the steps in exported pipeline
    set_param_recursive(mypipeline.steps, 'random_state', 42)
    mypipeline.fit(X_train, y_train)
    return mypipeline

def select_features(trained_model, X_train, X_test):
    model = SelectFromModel(trained_model, prefit=True)
    # Create new dataframes with selected features
    model.set_output(transform="pandas")
    X_train = model.transform(X_train)
    X_test = model.transform(X_test)
    # Drop these three features from the selected nine features
    X_train.drop(['starttime_T2','endtime_T1','Disturbances.section'],axis=1,inplace=True)
    X_test.drop(['starttime_T2','endtime_T1','Disturbances.section'],axis=1,inplace=True)
    #print(X_train, X_test)
    return X_train, X_test

tpot_data = pd.read_csv('matrix_data.csv', sep=',', dtype=np.float32)
features = tpot_data.drop('target', axis=1)
```

```
34
35 X_train, X_test, y_train, y_test = \
36             train_test_split(features, tpot_data['target'], random_state=42)
37
38 print("Number of classes =", y_train.nunique())
39 init_model = train_model(X_train, y_train, "rf")
40 print("Accuracy =", accuracy_score(y_test, init_model.predict(X_test)))
41
42 X_train, X_test = select_features(init_model['randomforestclassifier'], X_train, X_test)
43 # Train a new model with the selected features
44 final_model = train_model(X_train, y_train, "rf")
45 print("Accuracy = ", accuracy_score(y_test, final_model.predict(X_test)))
```

References

Andriy Burkov. *Machine learning engineering*, volume 1. True Positive Incorporated Montreal, QC, Canada, 2020.

Selim Dündar and İsmail Şahin. Train re-scheduling with genetic algorithms and artificial neural networks for single-track railways. *Transportation Research Part C: Emerging Technologies*, 27: 1–15, 2013. doi:10.1016/j.trc.2012.11.001.

Mohammad Hossein Fazel Zarandi, Ali Akbar Sadat Asl, Shahabeddin Sotudian, and Oscar Castillo. A state of the art review of intelligent scheduling. *Artificial Intelligence Review*, 53:501–593, 2020. doi:10.1007/s10462-018-9667-6.

Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.* " O'Reilly Media, Inc.", 2022.

Google. Machine learning glossary. *Available: https://developers.google.com/machine-learning/glossary*, 2017.

Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212, 2021. doi:10.1016/j.knosys.2020.106622.

Sai Prashanth Josyula. *Parallel algorithms for solving the train timetable rescheduling problem.* PhD thesis, Blekinge Institute of Technology, Department of Computer Science, 2021.

James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pages 1–10, 2015. doi:10.1109/DSAA.2015.7344858.

Maryam Karimi-Mamaghan, Mehrdad Mohammadi, Patrick Meyer, Amir Mohammad Karimi-Mamaghan, and El-Ghazali Talbi. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, 296(2):393–422, 2022. doi:10.1016/j.ejor.2021.04.032.

P. Kuppusamy, S. Venkatraman, C.A. Rishikeshan, and Y.C.A. Padmanabha Reddy. Deep learning based energy efficient optimal timetable rescheduling model for intelligent metro transportation systems. *Physical Communication*, 42:101131, 2020. doi:10.1016/j.phycom.2020.101131.

Jie Li, Zhongcan Li, Chao Wen, Qiyuan Peng, and Ping Huang. Train operation conflict detection for high-speed railways: a naïve bayes approach. *International Journal of Rail Transportation*, pages 1–19, 2022. doi:10.1080/23248378.2022.2071346.

Shiyun Li, Tianzong Yu, Xu Cao, Zhi Pei, Wenchao Yi, Yong Chen, and Ruifeng Lv. Machine learning-based scheduling: a bibliometric perspective. *IET Collaborative Intelligent Manufacturing*, 3(2):131–146, 2021. doi:10.1049/cim2.12004.

Lingbin Ning, Yidong Li, Min Zhou, Haifeng Song, and Hairong Dong. A deep reinforcement learning approach to high-speed train timetable rescheduling under disturbances. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3469–3474, 2019. doi:10.1109/ITSC.2019.8917180.

Mitsuaki Obara, Takehiro Kashiyama, and Yoshihide Sekimoto. Deep reinforcement learning approach for train rescheduling utilizing graph theory. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4525–4533, 2018. doi:10.1109/BigData.2018.8622214.

Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR, 2016.

Md Siddiqur Rahman, Laurent Lapasset, and Josiane Mothe. Multi-label classification of aircraft heading changes using neural network to resolve conflicts. In *ICAART (3)*, pages 403–411, 2022.

Ruifan Tang, Lorenzo De Donato, Nikola Besinović, Francesco Flammini, Rob M.P. Goverde, Zhiyuan Lin, Ronghui Liu, Tianli Tang, Valeria Vittorini, and Ziyulong Wang. A literature review of artificial intelligence applications in railway systems. *Transportation Research Part C: Emerging Technologies*, 140:103679, 2022. doi:10.1016/j.trc.2022.103679.

Zhuang Wang, Weijun Pan, Hui Li, Xuan Wang, and Qinghai Zuo. Review of deep reinforcement learning approaches for conflict resolution in air traffic control. *Aerospace*, 9(6):294, 2022.

Chao Wen, Ping Huang, Zhongcan Li, Javad Lessan, Liping Fu, Chaozhe Jiang, and Xinyue Xu. Train dispatching management with data-driven approaches: A comprehensive review and appraisal. *IEEE Access*, 7:114547–114571, 2019.

Marcel Wever, Alexander Tornede, Felix Mohr, and Eyke Hüllermeier. Automl for multi-label classification: Overview and empirical evaluation. *IEEE transactions on pattern analysis and machine intelligence*, 43(9):3037–3054, 2021.

Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

Jiamin Zhang, Jiarui Zhang, et al. Artificial intelligence applied on traffic planning and management for rail transport: A review and perspective. *Discrete Dynamics in Nature and Society*, 2023.

Marc-André Zöller and Marco F Huber. Benchmark and survey of automated machine learning frameworks. *Journal of artificial intelligence research*, 70:409–472, 2021.