**Predicting Medium-Term Train Delays**

**Dominic White**

**Noura Al-Mouyabed**

**Abstract**

1. **Context / background**

Trains are one of the most popular forms of public transport. The perceived likelihood and severity of train delays is a turn-off for many consumers, and so a system predicting train delays in the medium-term would improve both customer satisfaction and operating profits. There are currently no solutions to this problem in the literature, nor in practice, although research has been carried out into real-time delay prediction.

1. **Aims**

The aims of this project are twofold. The primary aim is to identify a suitable model for predicting medium-term train delays. The secondary, supporting aim is to develop a new train delay dataset to train this model on, as there are none publicly available.

1. **Method**

Using historical schedule data, movement data, location data, and weather data from a variety of sources, a dataset is constructed for use in machine learning. A variety of classification and regression models are trained, tested, and optimised. The effect of including weather data, and various encoding schemes, is investigated.

1. **Results**

An entirely new dataset has been constructed. It is sufficiently robust for general usage, although there remain discrepancies. Of the classification models tested, a Random Forest achieved the greatest -measure of . The best performing regression model was also a Random Forest, with an MSE of .

1. **Conclusions**

These solutions, though insufficient practical, form a solid foundation for future work in this area. The inclusion of more exogenous data would likely have further improved results.

1. **Keywords**

Machine learning, train delay, delay prediction, extract-transform-load, etl, classification, regression

**I. INTRODUCTION**

1.76 billion rail journeys were made in the 2018 - 2019 financial year[[1]](#footnote-1). Of these journeys, approximately 12.2% were delayed by more than 5 minutes[[2]](#footnote-2). Train delays impose a huge cost on passengers and operators by contributing to the inefficiency of train operations (Van Oort, 2011). In 2006 – 07, for example, delays cost a minimum of £1 billion in terms of time lost to passengers (NAO, 2008). Furthermore, under a scheme known as ‘Delay Repay’, train operating companies (TOCs) are automatically obliged to refund 50% of the cost of passengers’ tickets if a train is between 30 – 60 minutes late and 100% if it is more than 60 minutes late. 4.6 million fares were repaid in 2018 - 2019.

So it is no understatement to say that train delay is a serious problem. The issue is compounded for passengers, who tend to perceive punctuality as worse than it is for three reasons: passengers tend to recall delayed trains over punctual trains; more passengers travel on late trains than punctual trains (as passengers accumulate while waiting); and operators avoid early running, so late trains are not balanced out by early trains (Harris and Godward, 1997, pg. 130). Furthermore, passengers expect trains to run on time as TOCs are considered to be in greater control of their environment (Murray, 1989, pg. 27-28).

This mismatch in perception places great importance on reducing the impact of delays on passengers. So: what if passengers could know in advance whether or not a train would be delayed? What if they could know *by how much* it would be delayed? The aim of this project is not to answer these questions, but to see whether doing so is possible: can train delays be predicted in advance?

Systems already exist for predicting real-time train delay; the problem is well-studied. The majority are analytical (Oneto et al, 2016). Such models are *online*, i.e. updated as information on train movements becomes periodically available. At the other end of the spectrum is the optimisation problem of train scheduling, which is also very well studied. Between the two extremes lies is a poorly-defined region, the *medium-term* timeframe, neither online nor ‘offline’, lower-bounded by the availability of forecasts of *exogenous data* and upper-bounded by the times trains depart their origins. This is the timeframe this project focuses on. Knowing that a train will be delayed while on it is of little help to most passengers, but knowing it will be a week in advance is.

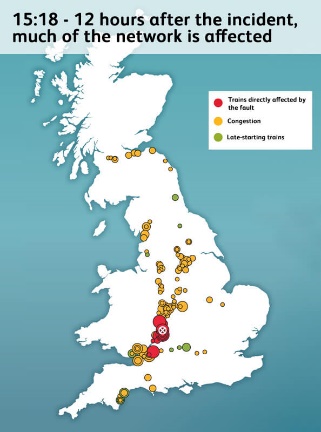
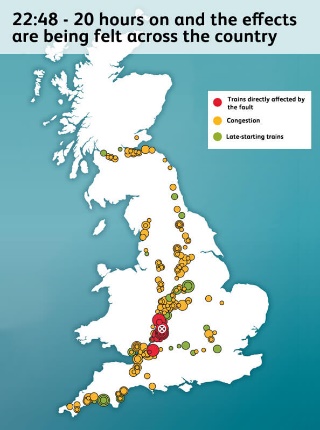
This project will explore data-driven models, which have recently been gaining traction in the field. RFI, the Italian railway manager, uses a (real-time) model developed by Oneto et al. (Oneto et al, 2016); Deutsche Bahn, a German railway operator, recently commissioned a comprehensive study into a (real-time) model suitable for their network (Lessen et al., 2019). This work aims to develop both a train delay dataset and two data-driven models – one classification, one regression – to see whether medium-term train delays can be predicted.

1. **Background**

Although precise terminology differs, the literature agrees that there are two principal classes of delay (Olsson, 2004): primary (exogenous) and secondary (knock-on, consecutive). A primary delay is caused by external stochastic disturbances (Oneto et al, 2016), of which there are many potential causes (Berger et al, 2011; Milinkovic et al, 2013):

* Signals and points failures
* Severe heat
* Fatalities
* Flooding
* Landslips
* Leaves
* Snow and ice
* Vandalism and trespass
* Accidents
* Malfunctioning equipment
* Engineering work
* Prolonged alighting and boarding times

These causes can be classified as due to weather, infrastructure, engineering, or passenger factors. Of the delays in 2006 - 2007, 42% were caused by infrastructure faults, 38% by TOCs, and the remaining 20% by events such as adverse weather, fatalities and vandalism.



A secondary delay is generated by operational conflict (Cerreto et al, 2016). Primary delays induce a cascade of secondary delays of other trains, which must wait for tracks to be clear, crews to be in the right place, platforms available, and so on. Predicting secondary delays is very difficult; it is the source of the complexity of real-time models. As Lessan *et al.* (Lessan *et al.,* 2019) note, some delay factors are predictable and controllable: most are neither. This project therefore focuses on predicting primary delays, which depends on *exogenous* data.

Data is *exogenous* if it is independent of other input data but the output is dependent upon it. In the context of primary delay prediction, the more of the causes of primary delay that can be incorporated into a model, the better it will perform. Models tend to use a combination of infrastructure (Markovic et al, 2015; Milinkovic et al, 2013; Lessan et al, 2019), weather (Oneto et al, 2016; Wang and Zhang, 2018; Oneto et al, 2017; Lessan et al, 2019), expert opinions (Markovic et al, 2015; Oneto et al, 2019) and engineering work (Lessan et al. 2019). Oneto et al. (2017) also recommends using information about passenger flows and about railway asset conditions. Where directly relevant exogenous data is not available, it is possible to use proxies (Harris, 1992), such as:

* train length (as a proxy for the number of doors to manage, and passenger demand)
* distance covered (as a proxy for the likelihood of encountering track defects and other technical / operational problems)
* the previous number of stops (as a proxy for cumulative delay resulting from passengers alighting and boarding)
* the age of the motive power unit (as a proxy for reliability; it is industry-held fact that a motive unit’s reliability declines after 20 years)
* track occupation (as a proxy for capacity utilisation the railway, and thus the likelihood of delays propagating)

1. **Objectives**

The research question proposed is: *can medium-term train delays be predicted?* To address this question, the objectives for this project were divided into three categories: minimum, intermediate, and advanced.

The minimum objective of this project is to construct a high-quality dataset incorporating a variety of exogenous data. The purpose of this objective is twofold. It is, first and foremost, a prerequisite for machine learning. Secondly, however, it will establish a strong understanding of the data itself, informing feature engineering in the intermediate and advanced objectives. There are few objective criteria for what makes a good dataset, save that it ought to be clean, logically structured, and complete, and fields included relevant.

The intermediate objective is to develop a binary classification model to predict *whether* a train will be delayed. A variety of classification models will be selected, with selection informed by a literature review, then tested. The best-performing model will further tuned. With no benchmark for comparison, any accuracy target would be arbitrary; so this model is expected to be better than a random guess.

The advanced objective is to build upon the intermediate objective by developing a regression model to predict *by how much* a train will be delayed. This problem is markedly harder than classification, and so performance is expected to be corresponding worse too. As before, the best-performing model will be extensively fine-tuned.

1. **Achievements**

TODO

**II. RELATED WORK**

Although there has been significant academic effort into producing *real-time* TDP systems, research on *medium-term* TDP is rather sparse. Fortunately, there is considerable overlap between the two timeframes. Solutions to the problem of real-time TDP use a variety of machine learning regression models, with the field tending towards ensembles / hybrids and random forests. The inclusion of exogenous data pertaining to weather and infrastructure is also popular. There are many classifications of TDP models, based on scope, model type, and solution methods (Markovic et al, 2015). Generally, approaches may be divided using two axes: whether they are offline or online, and whether they use analytical or data-driven models. As this project is concerned with latter, the former is discussed only briefly. Online models are updated with new data as it comes available; data-driven models are trained using historical data.

1. **Analytical approaches**

An analytical model is “primarily quantitative or computational in nature and represents the system in terms of a set of mathematical equations that specify parametric relationships and their associated parameter values as a function of time, space, and/or other system parameters” (Friedenthal and Steiner, 2015). Current state-of-the-art train delay prediction systems use analytical models (Oneto et al, 2016): “static rules, built by experts on railway infrastructure, and based on classical univariate statistics”.

Simplistic early models, such as those developed by Frank (1966) make overly restrictive assumptions about railway operations, e.g. no overtakes are allowed, departure times are uniformly distributed, and that the speed of each train is unique and constant. Subsequent work in this area has largely relaxed these assumptions: Peterson (1974), and Chen and Harker (1994), included factors such as overtakes, different speeds, priority systems, and uncertainties associated with train departure time. Further models have become increasingly advanced, incorporating stochastic approximation (Carey et al, 1994), and the impacts of dispatching strategies on train delays and passenger waiting time. The current state-of-the-art is likely Berger *et al.* (2011), whose model is currently used by Deutsche Bahn (DB), a German railway operator.

Much work has also been done by Kecman and Corman (Kecman, Corman, and Meng, 2015; Kecman, Corman, Peterson, and Joborn, 2015; Corman and Kecman; 2018) which blurs the line between analytical models and data-driven models. The authors use stochastic models based on Bayesian networks and trained on historical data to predict delays, with considerable success.

However, there is a fundamental limit on the complexity of such models, and as the field of machine learning has matured, the applicability of data-driven models to TDP has been increasingly well explored.

1. **Online / dynamic data-driven approaches**

Most real-time TDP systems are online. The first comprehensive attempt at developing a TDP system was made by Oneto *et al.* (2016). Subsequent papers (Oneto *et al.,* 2017; Oneto *et al.,* 2019) have expanded proposed models in scope and performance.

The authors consider a rail network a graph, where nodes represent checkpoints and edges tracks connecting them. A train follows an itinerary composed of checkpoints characterised by an origin, a terminus, and intermediate locations such as stops and transits. A schedule is modelled as a time series forecast problem, with performance at previous checkpoints used to predict performance at subsequent checkpoints. A model for each train is trained using historical delay data. The authors tested random forests (RF), extreme learning machines (ELM), and kernel methods (KM) and found that their RF performed twice as well as current state-of-the-art TDP systems. The inclusion of weather further improved accuracy by approximately 10%. Their model was very computationally expensive, however, requiring approximately 600,000 models to be trained daily across the entire rail network.

The authors then generalise their work to produce a dynamic data-driven TDP system (Oneto *et al.,* 2017), with performance tuned using *thresholdout*, which reduces overfitting. They compare the performance of two implementations of shallow and deep ELMs.

This work laid the groundwork for their most recent paper, which combines an experience-based model (EBM) and multiple RF into a hybrid model (HM) (Oneto *et al.,* 2019). The EBM uses operators’ knowledge and experience of the network to inform features. As Martin (2016) notes, in real-world train operations, delay prediction relies heavily on the experience and intuition of a local dispatcher, rather than a network-work computational instrument. The HM is a decision tree where each leaf is a RF. Trains are directed to the appropriate RF by similarity, eliminating the need for a model for every train. A new leaf is added each time a new train is seen that belongs to a previously unexplored branch of the decision tree. The RF regressor in the leaf is trained based on all the past train movements in that leaf. This model has several unique features. Trains older than 3 months are ‘forgotten’, to keep the size of the model constrained. Furthermore, this model naturally handles the mercurial nature of train schedules, which are both released periodically and revised constantly. The model required only 10 days’ of data after a new schedule comes into effect to reach optimal accuracy, with excellent performance also noted for outliers. The HM offers the best trade-off between accuracy and computational requirements, with superior results to all current models.

A similarly impressive model was developed by Lessan *et al.* (Lessan *et al.,* 2019), working closely with Deutsche Bahn (DB). The authors used 3.25 years’ of data from DB and incorporated a wide range of exogenous data, with approximately 350 features for operational (i.e. currently running) trains and 70 for non-operational trains. The authors tested support vector regression (SVR) but settled on RF. Three models were used for operational trains: an online RF, mesoscopic simulation, and kernel regression. Two were used for non-operational trains: an offline RF and mesoscopic simulation. The authors found an accuracy of over 80% in predictions within a 60-minute horizon, a considerable improvement in real-time forecasts, but found that beyond that horizon predictions were only marginally better than the schedule.

1. **Offline / static data-driven approaches**

Static models are not updated with data as it comes available. Early attempts used simple neural networks (Peters et al., 2005) and were essentially proofs-of-concept. The use of neural networks was further investigated by Yaghini *et al.* (Yaghini et al., 2011) using data from Iranian Railways to predict the late arrival of passenger trains. Their model performed better to alternatives such as decision trees and logistic regression. The authors also tested the impact of various encoding schemes.

Subsequent work is largely exploratory, with no clear direction as described in the previous section. Papers investigate a wide variety of models, such as fuzzy Petri nets (FPNs) (Milinkovic *et al.,* 2013), support vector regression (SVM) (Markovic *et al.,* 2015), logistic regression (Wang and Work, 2015) and gradient-boosted random forests (Wang and Zhang, 2019).

FPNs are mathematical modelling tools used to analyse and simulate concurrent systems (Murata, 1989). The authors explored two separate FPNs. In the first, expert knowledge was used to define fuzzy sets and rules. In the second, an Adaptive Network Fuzzy Inference System (ANFIS) was trained on historical delay data and then replicated in an FPN. Both were then tested with real data from a Belgrade station node. The ANFIS-FPN produced results within 5% of actual delay values for a subset of the data; slightly worse performance was observed for the expert-defined FPN.

Markovic *et al*. (Markovic *et al.,* 2015) presented the first use of SVM. They found it outperformed artificial neural networks. Data for the analysis was again collected from Serbian Railways. The paper used the expert opinions of dispatchers to estimate the likelihood of multiple factors along a rail line, such as single-tracking, junctions, or the number of stations, causing a delay, using the Delphi method to obtain a final estimate. A strong correlation was found between expert opinions and train delays. The focus was on developing a functional relationship between delays and infrastructure so the effect of infrastructure improvements on delays can be predicted and valued.

Wang and Work (Wang and Work, 2015) used data from Amtrak to develop two vector autoregression models: one offline, the other online. The offline model improved the RMSE of predicted delays by 12%; the online model, by 60%. Vector autoregression is a model that capture the linear interdependence between multiple time series (i.e. each train).

Wang and Zhang (Wang and Zhang, 2019) present a relatively simple gradient-boosted regression trees (GBRT) model. The model was trained on a three-month dataset of weather, train delay, and train schedule records. Their model can make predictions up to 10 days’ in advance, the limit of the weather forecast system used, but performed poorly, which the authors attribute to the limited size of their dataset. This paper most closely resembles the objectives of this project.

**III. SOLUTION**

The solution designed to build a dataset, and predict medium-term train delays, is presented in this section. The Python programming language was used throughout, with NumPy for numerical operations, Pandas for data manipulation and analysis, scikit-learn for machine learning, and Matplotlib for graphing.

1. **Dataset construction: the Extract-Transform-Load (ETL) pipeline**

Central to the importance of machine learning is a high-quality labelled dataset. Unfortunately, there are no publicly available datasets for train delay. Correspondence between authors of previous papers (Yaghini (2011), and Wang and Zhang (2019)) proved unfruitful, so it was necessary to construct a new dataset, which became the minimum objective of this project. Dataset construction is a complex task. It falls under the purview of the Extract-Transform-Load (ETL) pipeline[[3]](#footnote-3). This pipeline *extracts* data from sources, *transforms* them into a usable format, and *loads* them into storage for future operations. As Ponniah (2010, pg. 284) sadly notes, “it is not uncommon for a project team to spend as much as 50% to 70% of project effort on ETL functions”. This proved the case with this project, so considerable space is dedicated to explaining each stage in the pipeline, and how it was realised in the context of this dissertation.

First, suitable data sources were identified for the four classes of primary delay (weather, infrastructure, passenger, and engineering) and the actual train data itself. The UK rail infrastructure manager Network Rail (NR) opened its feeds[[4]](#footnote-4) to developer usage in 2011.

|  |  |  |  |
| --- | --- | --- | --- |
| **Acronym** | **Description** | **Frequency** | **Relevance** |
| **BPLAN** | Train planning data, including locations and sectional running times. | Twice a year | I |
| **CORPUS** | Location reference data. | Monthly | U |
| **MOVEMENT** | Train positioning and movement event data. | Real-time | U |
| **RTPPM** | Performance of trains against the timetable, measured as the percentage of trains arriving at their destination on-time. | Once a minute | N |
| **SCHEDULE** | Daily extracts and updates of train schedules from the Integrated Train Planning System (ITPS). | Overnight each night | U |
| **SMART** | Train describer berth offset data used for train reporting. | Monthly | N |
| **TD** | Train positioning data at a signalling berth level. | Real-time | N |
| **TSR** | Details of temporary reductions in permissible speed across the rail network. | Once a week on Friday | I |
| **VSTP** | Train schedules created via the VSTP process | Real-time | I |

Table . Characteristics of the NR data feeds and their relevance to this project. I = investigated, U = used, N = not used.

**SCHEDULE**

Schedules are available as CIF files. A full CIF – a database snapshot – is released every Friday morning, with update CIF released every morning. Each update must be applied to the latest full CIF to maintain a correct database. Each file contains train schedules, metadata, associations, as well as changes to TIPLOCs.

**MOVEMENT: TD, TRUST, and Darwin**

The TD feed provides information about the position of trains through a network of *berths*. A berth usually represents a signal. TD was discarded as too low-level for this project. There are two movement systems in use: Train Running Under System TOPS (TRUST) and Darwin. TRUST is a NR system used for monitoring the progress of trains and tracking delays in the UK. Darwin uses both TRUST and TD for real-time data, and also incorporates Darwin workstations, Customer Information Systems (CIS), and internal messaging systems.

TD feeds into TRUST, and TRUST into DARWIN. Darwin provides more comprehensive information. While the objectives of this project were being defined, it was undecided whether to focus on real-time or medium-term train delays. Real-time would have necessitated the use of Darwin, where medium-term could have used either. To keep options open, Darwin was therefore used. Darwin messages contain a lot of extraneous information, but fundamentally each movement message contains a timestamp and a location, either a STANOX or TIPLOC. Both CORPUS and NaPTAN are therefore necessary to ensure a message can be geolocated.

**Location: CORPUS (and NaPTAN)**

CORPUS is used in conjunction with the National Public Transport Access Node (NaPTAN) database, a nationwide system for uniquely identifying all points of access (nodes) to public transport in the UK. Only rail stations are, naturally, of concern here. A frankly alarming number of codes are used to refer to locations in the UK rail network; they are described below, along with which datasets support them.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **CORPUS** | **NaPTAN** | **Description** |
| **STANOX** | Y |  | Station Number. First two digits are the geographic area. Can refer to non-station locations such as sidings and junctions. Numbers run broadly north-to-south. |
| **UIC** | Y |  |  |
| **CRS / NRS / 3ALPHA** | Y | Y | 3-character limit. Used primarily to identify stations and on seat reservation labels. |
| **TIPLOC** | Y | Y | Timing Point Location. Relates to points used in deriving train schedules. 7-character limit. A station often has multiple TIPLOCs if it consists of multiple groups of platforms on different lines. |
| **NLC** | Y |  | National Location Code. 6-character limit. Identifies locations on the railway. Used for retailing and accounting purposes. |
| **NLCDESC** | Y |  | A description of the NLC. No limit. |
| **NLCDESC16** | Y |  | A description of the NLC. 16-character limit. |
| **ATCO** |  | Y |  |
| **EASTING, NORTHING** |  | Y | Geographic Cartesian coordinates for a point. Uses EPSG:27700[[5]](#footnote-5), the British National Grid / Ordnance Survey system. |
| **NAME** |  | Y | The name of the location. |

**Weather: CEDA, MIDAS, and Datapoint**

The primary source of weather data is the Met Office Integrated Data Archive System (MIDAS). MIDAS is a database of land and marine surface observations, collected from 1853 to the present day, by the Met Office station network. MIDAS offers several datasets. The most comprehensive is the Hourly Weather Observation Data, which contains meteorological values measured on an hourly time scale. These observations include 104 fields[[6]](#footnote-6), though many are for quality control, or too specific to necessitate inclusion. Station data is available from the Centre for Environmental Data Analysis (CEDA). Each station is geolocated by latitude and longitude.

For this project to be useable, trained models must be applicable to unseen data. Unseen train data are simply train schedules. Unseen weather data are *forecasts*. Forecasts are available from Met Office Datapoint, a service allow access to freely available Met Office data feeds. They may be obtained as 3-hourly site-specific forecasts up to 5 days’ in advance. There are over 5,000 UK forecast sites. There are 10 forecast available fields.

**Infrastructure: BPLAN and the Train Planning Network Model**

The Train Planning Network Model is used by the Integrated Train Planning System (ITPS). Unfortunately, integrating either of these datasets proved infeasible; there was simply too little documentation, and so errors during parsing could not be fixed.

**Engineering: TSR**

TSR provides proxy details of engineering works. It was unfortunately not archived in the repository used for the rest of the NR data, and so could not be included.

1. **Extraction**

Extraction is the process of collecting data, often from multiple different sources, and moving it to a *staging area.* Some basic validation also takes place at this stage.

|  |  |
| --- | --- |
| **Data class** | **Notes** |
| Schedule | Schedule data is available from Peter Hicks’ website[[7]](#footnote-7). Both full and update extracts are downloaded as gzipped files. |
| Movement | Darwin data is also available from Peter Hicks’ website[[8]](#footnote-8). Each day is a bzip2-compressed tar file containing 1440 files, one for each minute in the day. Each file comprises XML messages. Each minute is extracted in memory and written to one CSV for each day / file. |
| Location | NaPTAN is available as a zipped CSV from the Department for Transport (DfT). Only the railway nodes are extracted and saved. CORPUS is available from NR as a gzipped JSON. Only the TIPLOC information is extracted and saved. |
| Weather | This data is available for download as a CSV file, via FTP from CEDA, though headers must be downloaded separately. Each row has a *src\_id*, the identifier of the station responsible for that record. Station data is also available from CEDA as a KMZ file. A KMZ file is a zipped KML (Keyhole Markup Language) file; KML is an XML notation for expressing geographic annotations and visualisations. The KML file is parsed using pykml and station metadata using BeautifulSoup to produce a CSV. Some processing is done here: latitudes and longitudes are converted to floats, station IDs to integers, and headers to standard Pythonic form. |

1. **Transform**

Transformation is the process of applying a set of functions to extracted data, to clean, map, validate, and consolidate datasets, with the aim of making the data conform to a uniform schema. Typical operations might be mapping NULL to 0, formatting dates, removing duplicate records, converting units, splitting columns into multiple columns (or vice versa), and joining datasets. There is a lot of overlap between transformation and *pre-processing*, the stage just prior to training. The difference lies in ease of repetition: transformation would, ideally, be performed once, as operations are typically expensive, operating on vast quantities of unstructured and semi-structured data. Pre-processing, using heavily optimised code in NumPy and Pandas, can be iteratively improved.

Transforming schedules is a difficult task. To retrieve the correct schedule for a given UID on a given day, all schedules with that UID must be retrieved. Those not active on the *day* in question – as defined by *days\_run* – can be discarded, as well as those with a start date in the future. From those remaining, the schedule with the lowest *stp\_indicator* is the correct one, in order “C”, “O”, “V”, “P”. “C” is a planned cancellation; “O” is an overlay from STP; “V” is an overlay from LTP; “P” is the permanent base schedule.

As mentioned, an update CIF is released each morning. It contains deletions, amendments, and new schedules, which must be applied to the currently-held schedule database. After each update is applied, the schedules for that day are written to file by filtering those by the criteria above. There is tremendous scope for error. Train scheduling is a very complex problem, and this is reflected in the data formats. Fortunately, full schedules are available weekly, so errors are reset every week. It is likely the bugs in this stage will lead to later problems.

A CIF file contains 9 types of record: HX (header), BS (schedule metadata), BX (extended schedule metadata), LO (origin), LI (intermediate), LT (terminus), CR (change en route), AA (association) and ZZ (end of file). Of interest are BS, BX, LO, LI, LT. For each BS, a new record is created. A BX may be used to add additional metadata. Each subsequent point (LO, LI, and LT) is used to add new data to the record: the origin, time of departure, number of stops, for instance.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **LO** | **LI** | **LT** | **Description** |
| *type* | Y | Y | Y | Point type. Either LO (origin), LI (intermediate), or LT (terminus) |
| *location* | Y | Y | Y | Location. A TIPLOC |
| *suffix* | Y | Y | Y | The number of times the TIPLOC has appeared |
| *sta* |  | Y | Y | Scheduled time of arrival |
| *std* | Y | Y |  | Scheduled time of departure |
| *stp* |  | Y |  | Scheduled time of pass |
| *pta* |  | Y | Y | Planned time of arrival |
| *ptd* | Y | Y |  | Planned time of departure |
| *pass* | Y | Y | Y | Whether or not the event is a pass |
| *platform* | Y | Y | Y | The platform used by the service |
| *line* | Y | Y |  | Line to be used on departure from the location |
| *path* |  | Y | Y | Line to be used on arrival at the location |
| *activity* | Y | Y | Y | The activities occurring at this location |
| *engineering\_allowance* | Y | Y |  | Time allowed for recovery from engineering activities |
| *pathing\_allowance* | Y | Y |  | Time allowed for pathing requirements |
| *performance\_allowance* | Y | Y |  |  |

Two types of record are ignored to reduce complexity: CR (change en route) and AA (associations). An association represents some *dependency* between two trains, such as crew or engines. They have the potential to be a rich source of data: is, for instance, a train with more associations more likely to be delayed? Using AAs would necessitate a much more complex model – one that could handle dependencies between trains – rather than the (simpler) tabular format desired. CRs indicate that some metadata contained with BSs changes during a train’s journey. The metadata for a day’s schedules is roughly 60 MB. The *route* information for a day is roughly 800 MB. Storing the metadata once, rather than merging it onto every location record, significantly reduced space requirements. The other potential solution – checking a database of CRs for every L - would have incurred significant cost later in the pipeline.

**Transforming weather**

As mentioned previously, there is a need to map from the MIDAS format to DataPoint format. There are 10 available fields for forecasts, of which only 7 could be meaningfully mapped to equivalent MIDAS fields. For several, this was a simple unit conversion. Boolean masks are used to convert *wind\_speed* and *wind\_gust­­* to mph. For visibility and weather type, predefined codes were used to establish a map between the two formats[[9]](#footnote-9).

|  |  |  |
| --- | --- | --- |
| **MIDAS** | **Datapoint** | **Notes** |
| *prst\_wx\_id* | Weather type | MIDAS code definition depends on *src\_opr\_type.* Use mapping between relevant table and Datapoint code definition |
| *rltv\_hum* (%) | Screen relative humidity (%) | No conversion necessary |
| *visibility* (decametre) | Visibility | Used code definition |
| *air­­\_temperature* (°C) | Screen temperature (°C) | No conversion necessary |
| *wind\_direction* (degrees) | Wind direction (16-point compass) | Map degrees to compass directions, with 0 as N. |
| *wind\_speed* (knots or ms-1) | Wind speed (mph) | MIDAS unit depends on *wind\_speed\_unit\_id* |
| *q10mnt\_mxgst\_spd* (knots or ms-1) | Wind gust (mph) | MIDAS unit depends on *wind\_speed\_unit\_id* |
|  | Feels-like temperature (°C) | No equivalent |
|  | Precipitation probability (%) | Clearly nonsensical |
|  | UV Index | No equivalent |

**Transforming movement**

Each day of train movements is a file roughly 1.5GB in size. This file also contains a variety of other messages: forecasts, schedules, station messages, alarms, warnings, and so on. The data must be converted into a CSV for usage. It is assumed that no trains runs longer than 24 hours, i.e. there is no train that starts on a given date and ends two days afterward. Each record may have a selection of arrival, departure, and pass times: working, estimated, and public, as well as a lot metadata relating to whether to *show* certain fields to the public which can be safely discarded. Each train’s information is written to the date it started, and ultimately comprised just five columns: service start date, UID, location, type, and time. It is common for several different messages to refer to the same train movement, a result of Darwin re-reporting movements when updating forecasts. In these instances, duplicates are dropped. If times disagree, the average is taken.

**Transforming location**

This stage was relatively simple. NaPTAN and CORPUS were merged, and the closest geographical weather station used in the dataset was calculated for future weather look-up.

1. **Load**

Load is the process of storing data in a format accessible for future usage. It was a more complex in this project, involving the merging of the four data classes. Each row should be a train. For every UID in the schedule dataset for a given day, find the matching movement records, append the origin and terminus of each train, and calculate the delay using the value in the schedule values. Finally, use the origin and destination locations to identify the closest weather station, and from there add information on the weather experienced at the start and end of a train’s journey.

There is plenty of scope for error in this stage, in particular from ‘schedule’ UIDs not in ‘movement’ and mismatching TIPLOCs between the ‘schedule’ and ‘movement’ dataframes. These issues cannot be resolved. They are believed to be the result of two planning processes which generate, or modify, schedules: VSTP and STP. In short, it is possible there exists trains in ‘movement’ with no corresponding metadata in ‘schedule’, corresponding to schedules created by VSTP and STP, and vice versa, corresponding to schedules cancelled by either of the two.

**IV. RESULTS**

In this section the results of the classification and regression models are presented. The metrics used are also discussed. The experiment environment system specification is outlined below.

|  |  |
| --- | --- |
| **Component** | **Details** |
| Processor | Intel Core i5-6990K 3.6 GHz |
| GPU | NVIDIA GeForce 980 TI 8GB |
| RAM | 32GB 2400Mz DIMM |
| OS | Windows 10 |

For both classification and regression problems, a number of models were selected from *scikit-learn* and trained on the same 70% of the dataset using default parameters. The training dataset consisted of approximately 3.85 million rows and 299 columns. No limits were imposed on memory, CPU usage, or time. The models were then tested on the remaining 30% of the dataset, a number of metrics computed. Finally, the best-performing model was subject to hyperparameter tuning to further improve performance.

1. **ETL**

Pre-processing is necessary before using the data output from the LOAD stage. This mostly takes the form of encoding categorical variables, dropping unnecessary columns, and setting datatypes to save space.

Special attention is paid to datetimes, as they are likely to have significant influence. Dates are split into their respective parts: year, month, day, day of week. Hour and minutes are encoded into a single minutes variable. Different variables are used for both origin and destination. These are encoded cyclically[[10]](#footnote-10) by converting a single column to two: one for and the other for . In this way a variable is encoded as a unit circle, so Monday and Sunday are adjacent, for instance, as are 11pm (23 hours) and 12am (0 hours).

Two target columns are then constructed using the difference between the scheduled arrival time (STA) and actual arrival time (ATA). The first, whether or not a train is delayed, is used for classification. The second, the amount by which a train is delayed, is used for regression. Erroneous values are also discarded. In about 1% of records, an error in the ETL pipeline resulted in ATAs a day *before* STAs. This occurred on a small number of trains that run over two days, which were therefore discarded.

Some element of locality is also likely to improve results. Categorically encoding origin and destination TIPLOCs is impractical, as there are approximately 80,000. Nor is there any structure to their designations. Instead, their corresponding STANOX area is used. There are 89 STANOX areas[[11]](#footnote-11), proxies for geographical areas.

Dropped columns are mostly IDs, and those used to construct the columns discussed above. The actual time of departure (ATD) must also be dropped, as it could not be known in real-life data ahead of time. Also dropped are three metadata columns with too many NaN values: *sleepers*, *reservations*, and *branding*.

Comparable datasets are much smaller in size. Yaghini *et al.* (Yaghini *et al*., 2011) used 6 columns: an origin-destination pair, the rail corridor, the day, the month, and the year; Wang and Zhang (Wang and Zhang, 2019) used 8; Wang and Work (Wang and Work, 2015) 6 columns. It is believed that this should result in superior results.

1. **Binary classification**

In the dataset, approximately 10% of rows are positive (i.e. delayed). This gives rise to misleading accuracies: a model can simply always predict negative (i.e. not delayed) to achieve approximately 90% accuracy. This is known as the accuracy paradox. So three other metrics are used: precision, recall, and the -measure. The -measure is the weighted harmonic mean of precision and recall, which captures the natural trade-off between them.

Seven classifiers were chosen for the initial round of testing: gradient-boosting, histogram-gradient-boosting, linear SVC, logistic regression, random forest, and ridge.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | TPR | TNR | Acc | Prec | Rec |  |
| AdaBoostClassifier | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| DecisionTreeClassifier | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| GaussianNB | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| GradientBoostingClassifier | 0.0 | 0.0 | 0.90 | 0.59 | 0.02 | 0.04 |
| HistGradientBoostingClassifier | 0.0 | 0.0 | 0.90 | **0.68** | 0.04 | 0.07 |
| LinearSVC | 0.0 | 0.0 | 0.90 | 0.00 | 0.00 | 0.00 |
| LogisticRegression | 0.0 | 0.0 | 0.90 | 0.45 | 0.02 | 0.05 |
| MLPClassifier | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| RandomForestClassifier | 0.0 | 0.0 | 0.90 | 0.58 | **0.22** | **0.32** |
| RidgeClassifier | 0.0 | 0.0 | 0.90 | 0.49 | 0.01 | 0.01 |
| SGDClassifier | 0.0 | 0.0 | 0.90 | 0.00 | 0.00 | 0.00 |

Each model trivially achieved 90% accuracy. As expected, models which try to optimise a linear function – SGD, and LinearSVC – failed to converge. The table above shows that the RandomForestClassifier is the best model for solving this binary classification problem, with by far the greatest recall and -measure, though the results are still disappointing.

Subsequent experiments with the RF found that the different encoding schemes previously discussed had little impact. Furthermore, RFs offer few opportunities for hyperparameter tweaking, with the exception of increasing the number of trees in the forest (*n\_estimators*). With the default value of 100, the RF took approximately an hour to train.

A recall of 0.22 means that the model correctly identified 22% of ‘delay’ occurrences. A precision of 0.68 means that 68% of ‘delay’ identifications were correct.

$k$-fold

Cross-validation is a sampling procedure used to evaluate machine learning models on a limited data simple. $k$ is the number of groups that a given data sample is to be split into.

Even with the dataset tuned to the RF, resuts are still disappointing. We therefore investigated techniques to over- and under-sample the dataset to increase the proportion of positive (i.e. delayed) rows.

Some features do not have a clear distinction between categorical and one-hot encoding.

To evaluate the model, we performed repeated stratified $k$-fold validation, using the \verb|ROC\_AUC| metric.

The AUC - ROC curve is a

ROC is a probability curve and AUC represents the degree of separability. It is a measure of how capable a model is of distinguishing between classes. The ideal AUC - ROC curve appears logarithmic, and denotes the ability to perfectly distinguish between the positive and negative class.

**What is a random forest?**

A random forest (RF) is an ensemble of decision trees. Each decision tree is built on a random subset of the original data. At each tree node, a subset of features are randomly selected to generate the best split.

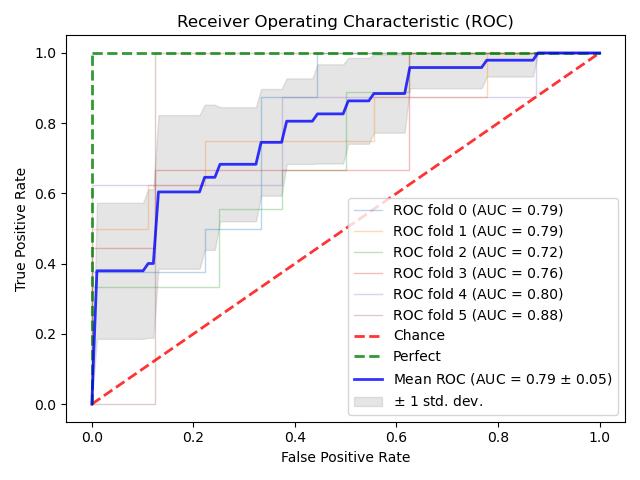
Random forests offer many advantages. They are

**Re-sampling**

The class imbalance is a major issue. To resolve this, we oversampled the minority class (i.e. delayed) using SMOTE (Chawla et al., 2002) and randomly undersampled the majority class (i.e. not delayed) to achieve a 50:50 ratio. SMOTE interpolates between minority instances nearby in the feature space to produce synthetic minority instances.

**Advanced evaluation**

To better evaluate the model, we use repeated stratified 10-fold cross-validation with the ROC-AUC metric. This is “generally a better scheme, both in terms of bias and variance, when compared to regular cross-validation” (Kohavi, 1995), and selects folds so that each contains roughly the same class distribution in the original dataset (in this case, the re-sampled dataset).



ROC curves visualise the trade-off between the true positive rate (TPR) and false positive rate (FPR) for a predictive model using different probability thresholds. The top-left corner of the plot is the ‘ideal’ point: a false positive rate of 0.0, and a true positive rate of 1.0. A larger area under the curve (AUC) is therefore better. The ‘steepness’ of the ROC curve is also important, as it is ideal to maximise the true positive.

Could use GridSearch CV for this purpose. Yeah, let’s do that first. Hmm…

Max depth and number of estimators, basically.

Increasing the number of features increasing training time quadratically.

We may use Bayesian optimisation to find ot

1. **Regression**

Eight models were chosen for the initial round of testing. The majority are the regression counterparts of the models using in A, with the exceptions of the inclusion of Bayesian and Kernel Ridge, and the exclusion of Logistic Regression.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | MAE | MSE | RMSE |  | RMSLE |
| BayesianRidge |  |  |  |  |  |
| GradientBoostingRegressor |  |  |  |  |  |
| HistGradientBoostingRegressor |  |  |  |  |  |
| KernelRidge |  |  |  |  |  |
| LinearSVR |  |  |  |  |  |
| RandomForestRegressor |  |  |  |  |  |
| Ridge |  |  |  |  |  |
| SGDRegressor |  |  |  |  |  |

**V. EVALUATION**

In this section, we evaluate the strengths and weakness of our solutions, reflecting upon the original research question: *Can medium-term train delays be predicted?*. The approach to dataset construction is also discussed.

1. **Dataset construction**

Our approach to constructing a dataset was haphazard at best. Railways are a very complex subject matter, and with no expert at hand, this complexity was an intimidating prospect.

It took some time to settle on the research question. Was it better to tread the better-worn path of real-time delay prediction, and accept the enormous overhead of modelling a railway network, or to explore medium-term prediction?

This dilemma is represented in the movement dataset used. TRUST is briefly mentioned as an alternative, simpler, system to Darwin. Using it would have precluded the use of forecasts. Had this project pursued real-time delays, a key performance indicator would have been a comparison between the forecast made by the proposed system and those made by Darwin. By the time we had chosen to explore medium-term delays, it was too late to revert back to TRUST, and so greater overhead was incurred.

As discussed previously, predicting primary delays is dependent on the inclusion of exogenous data. Many of the relevant categories were unfortunately unavailable in the archive used and so could not be included. As this dissertation progressed, a new repository, archiving all feeds, was established by OpenTrainTimes[[12]](#footnote-12). However, switching would have incurred too much work, both from an extraction, but particularly a transformation, perspective, at a late stage of the project.

1. **Solutions**

Many modifications have been suggested to SMOTE since it was first developed. Using one of these may have yielded better results.

In the early stages of the project decisions were made…

These minor shortcomings in the planning of the project did not, however, have an adverse effect on the quality of the solution produced.

Damn. It just feels like way too much! Let’s go back to the code and try to calm down a little, hmm?

Possible source of error: No, they have timestamps. I could look there if I cared enough. And perhaps I do.

**VI. CONCLUSIONS**

In this project we have produced two solutions to the problem of medium-term train delay prediction, one classification, the other regression. To achieve this, a wide variety of models were tested, and their performance evaluated, with the best-performing models explored in great depth to further improve performance.

A brand-new dataset was also constructed. Though of reasonable quality, there remain some issues that need to be resolved before we deem it suitable for general use. Making such a dataset publicly available would likely spur further interest in this field, which would in of itself be an important contribution.

Future work would include more exogenous data. The repository mentioned in V. archives all NR feeds.

To make this project directly useful for consumers, a system could be developed that subscribes to real-time schedule and weather data to predict live whether a given train will be delayed, and, if so, by how much.

**VII. REFERENCES**

TODO

1. https://dataportal.orr.gov.uk/statistics/usage/passenger-rail-usage/ [↑](#footnote-ref-1)
2. https://www.finder.com/uk/train-statistics [↑](#footnote-ref-2)
3. Sometimes a different order is used: Extract-Load-Transform (ELT). The sole difference is where the transformation stage takes place. The distinction is largely academic here, so ETL is used throughout. [↑](#footnote-ref-3)
4. https://www.networkrail.co.uk/who-we-are/transparency-and-ethics/transparency/open-data-feeds/ [↑](#footnote-ref-4)
5. https://epsg.io/27700 [↑](#footnote-ref-5)
6. https://artefacts.ceda.ac.uk/badc\_datadocs/ukmo-midas/WH\_Table.html [↑](#footnote-ref-6)
7. https://cdn.area51.onl/archive/rail/timetable/index.html [↑](#footnote-ref-7)
8. https://cdn.area51.onl/archive/rail/darwin/index.html [↑](#footnote-ref-8)
9. https://www.metoffice.gov.uk/services/data/datapoint/code-definitions [↑](#footnote-ref-9)
10. https://ianlondon.github.io/blog/encoding-cyclical-features-24hour-time/ [↑](#footnote-ref-10)
11. https://wiki.openraildata.com//index.php?title=STANOX\_Areas [↑](#footnote-ref-11)
12. https://networkrail.opendata.opentraintimes.com/ [↑](#footnote-ref-12)