Australasian Transport Research Forum 2022 Proceedings
28-30 September, Adelaide, Australia
Publication website: http://www.atrf.info

Ridespace: Predicting short-term rail passenger crowding using Machine Learning

Richard Rossmann[1], Saul Gare[1], Sam Pardy[1]

[1]Department of Transport, 1 Spring Street, Melbourne 3000

Email for correspondence: sam.pardy@transport.vic.gov.au; saul.gare@transport.vic.gov.au

Abstract

Public transport users' preferences to avoiding crowded public transit services is increasingly becoming a concern for governments as we continue to transition into a post pandemic world. The impact of high crowding levels at rail stations and inside trains on waiting time, perceived travel time, and passenger wellbeing is of significance, especially during and post Covid-19 pandemic. Considering the influence of on-board crowding on travel behaviour and the importance of providing users with more information on public transit service occupancy levels, the Department of Transport in Victoria, Australia launched the Ridespace initiative in 2021, which provides estimated service occupancy for all metropolitan rail services across the next two operating days, as well as platform and station crowding for the current operating day. To achieve this, a deep neural network machine learning model was implemented and trained on historical patronage data. The model uses real-time timetable changes such as service cancellations and delays to update predictions throughout the day. A critical requirement of the model was to prevent underpredicting of crowding levels, so an additional cost matrix is applied to adjust quantile forecasts during the inference stage. This paper will discuss in detail how this methodology was applied in the context of near real-time public transport occupancy forecasting, what input datasets were used (both historical datasets for training, as well as real-time network data to update the predictions), and how the obtained results were validated.

1. Introduction

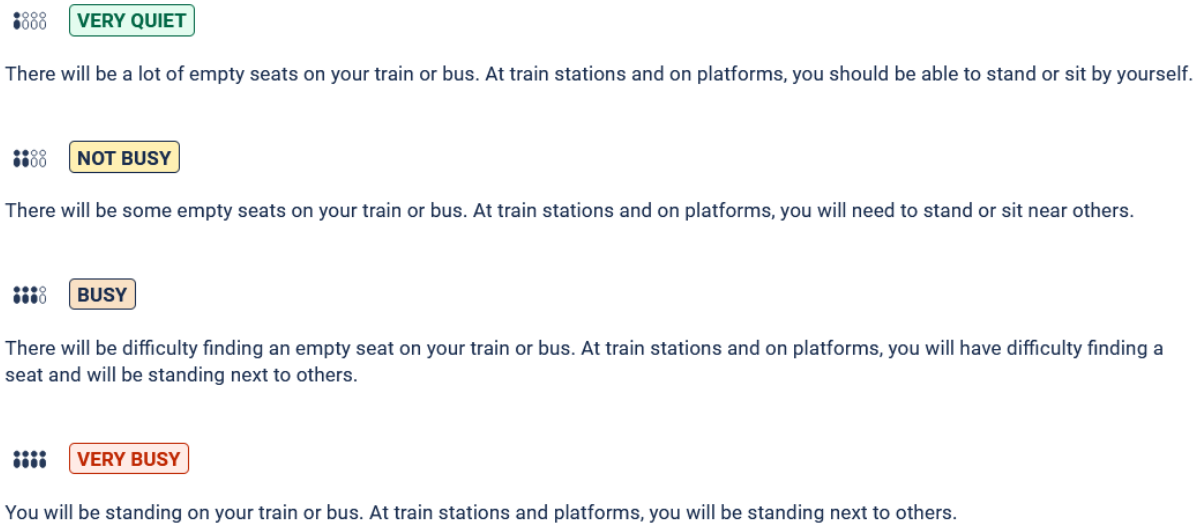
This paper first outlines the datasets used in section *2. Datasets*. The predictions are generated using a deep learning model based on (Lim et al. 2019)'s Temporal Fusion Transformers (TFT) architecture, which is discussed in *3. Methodology* and *4. Results*. We then briefly discuss some of the shortcomings and possible improvements, as well as how this work applies to the greater transport context in *5. Limitations* and *6. Applications*, respectively.

1.1 About Ridespace

A sharp decrease in public transport ridership has been observed during the Covid-19 pandemic around the world. Passengers are now likely to alter their behaviour with huge focus on key factors that contribute to the risk of transmission of Covid-19. Due to the unparalleled temporal and spatial scale of this catastrophe, some of the changes in travel behaviour may be continued even after a pandemic. To encourage a post Covid-19 restrictions return of patrons to the public transport (PT) network, the Department of Transport in Victoria (DoT) launched an initiative in 2021 to provide real-time and future predicted crowding at metropolitan rail stations, platforms and on train services. This initiative was successfully launched as "Ridespace" and features a web-based interface to surface predicted crowding levels as one of four categories: very quiet, not busy, busy, and very busy (see *Figure 1* for definitions).

The predictions are generated using an implementation of the TFT machine learning model for timeseries prediction. We utilized the open-source library PyTorch Forecasting (Beitner, 2020) to have access to the TFT architecture via a high-level python API, and to easily set up our datasets as a timeseries dataset class. Data processing, model training and real-time inference were all automated on the Databricks platform. See *3.2 Predicting Service Occupancy* for details of our real-time inference methodology.

Figure 1: Occupancy Level Definitions (DoT, 2021)



2. Datasets

The model was trained on several datasets, including historical public transport ticketing transactions with Myki cards, actual train arrival and departure times as well as headways between services, and the Train Service Usage Model (TrainSUM) – a statistical service assignment model developed by DoT to accurately estimate historical train boardings, alightings, transfers and service loads. More background on TrainSUM is provided in *Appendix*

A. The training data covered all metropolitan train stations and services from January 2018 to October 2021.

The TrainSUM data provides passenger loads by service and is regularised into an ordinary time series with 15-minute intervals by calculating the mean passenger load for services within a 30-minute window (15 minutes either side). This allows us to approximate typical service loads over the course of the day for each line and station combination.

Myki transactions (*ScanOns, ScanOffs*) were lagged by 1 day, and trainSUM data (*Average-*, *Log-* and *SquareDepartureLoad*, as well as *Average-*, *Log-* and *SquareLoadOn*) by 7 days. This was to simulate the actual inference vector, as the model would only have access to Myki data up to the day prior, and the TrainSUM model operates on a weekly schedule.

Table 1: Model Training Features

Static Categoricals	
StopName Direction Line	Grouping for each individual predictive timeseries. Direction is either inbound or outbound, while Stop and Line specify the spatial locations and connections.
Static Reals	
StopSequence (S)	Numerical value incremented by 1 for each stop between 1 and S_{max}, the number of stops for the service.
Time Varying Known Categoricals	
DayTypeCategory Month DayOfWeek Peak	Features to capture the periodicity and seasonality of PT patronage. Day Type includes normal weekday, weekend and public holiday; peaks are split across each 24 hour period into am, pm, inter and off peak.
Time Varying Known Reals	
TimeIdx (τ)	Index starting at 0 that is incremented by 1 for each time step
Year	Numerical year e.g. 2021
Headway*	Time in seconds since previous service at current stop
HeadwayLine*	Time in seconds since previous service with same headsign at current stop
SinTime* CosTime*	Cyclical time features to better capture day and night cycles, as shown in (Chakraborty & Elzarka 2019)
Disrupted*	Dummy variable to indicate occurrences of partial train line closures
Covid	Value between 0 and 4 to represent the current Covid-19 Restriction levels in Victoria
Time Varying Unknown Reals	
AverageDepartureLoad LogDepartureLoad SquareDepartureLoad	Derivations of Departure Load, the number of passengers on board the service at time of departure; lagged by 7 days.
AverageLoadOn LogLoadOn SquareLoadOn	Derivations of Load On, the number of passengers boarding the service at the current stop; lagged by 7 days
AverageLoadOff	Average number of passengers alighting the service at the current stop; lagged by 7 days
AvgDepartureLoadByStation AvgLoadOnByStation	Monthly historical average departure loads and boardings respectively at the current stop
ScanOns ScanOffs	Total hourly myki ticketing transactions at the current stops' ticket readers during the same hour on the previous day.

* Updated in real-time

3. Methodology

3.1 TFT Model

Temporal Fusion Transformer is a deep neural network (DNN), attention-based architecture for multi-horizon forecasting. It was introduced by (Lim et al. 2019). Our implementation follows the original closely, and we make no major changes to the proposed architecture. Our main contribution is successfully applying TFT in the domain of public transit passenger flow prediction.

Our implementation makes use of the Pytorch Forecasting (Beitner, 2020) high-level API to construct classes for our input dataset, training module and the TFT architecture itself. The parameters set for each class are specified in *Table 2*.

The input features specified in *Table 1* are transformed into a time series dataset class, which can be passed to a DNN model in batches during training. Data was grouped and normalized by *StopName*, *Direction*, and *Line*. The TFT treats the input timeseries as continuous steps in time, based on the time index specified. In our case however, each time step is one service departure for a particular stop, line and direction combination. This means we construct a separate timeseries for each group α, i.e. $\tau_a \in \{1, \dots, \tau_{a_{max}}\}$, where τ is the time index of each departure in chronological order.

We used a V100 16GB GPU for all our model training. The batch size was determined based on what would give us the best results without overloading the GPU during model fitting. As we want the model to always make predictions for an entire operating day, the maximum prediction length is set to 250, which is slightly more than the maximum number of service departures within any stop grouping observed in the training data. The maximum encoder length is similarly determined to cover 1 week of services (including on average five weekdays and two weekends with lower service frequencies).

Training was set up to run for a maximum of 200 epochs, however with an early stopping mechanism if the validation loss did not decrease across any 30-epoch training loop. Hyperparameters for gradient clipping, hidden size, dropout, hidden continuous size, attention head size and learning rate were optimized using Optuna (Akiba et al., 2019) hyperparameter tuning. For calculating losses, we used the same quantile loss as defined in (Lim et al. 2019), except instead of focusing only the 10^{th}, 50^{th} and 90^{th} quantiles, we generated output for all 100 quantiles between the 0^{th} and 99^{th} quantile: $Q \in \{0.00, \dots, 0.99\}$. The main purpose for this was so that we could reconstruct the probability density for each occupancy level category during post-processing, which is further explained in section 3.3.

Table 2: Data, Training and Model Class Parameters

Time Series Data Loaders	
Target Normalizer	GroupNormalizer
Groups	StopName, Direction, Line
Transformation	Softplus
Center output to 0	True
Eps (numerical stability)	1.0
Maximum Prediction Length	250
Maximum Encoder Length	1650
Batch Size	128

Pytorch Lightning Trainer	
Maximum Epochs	200
Gradient Clipping	0.4112*
Early Stop Callback	EarlyStopping
Monitor	Validation Loss
Minimum Delta	0.0001
Patience	30
Temporal Fusion Transformer	
Hidden Size	56
Dropout	0.2274*
Hidden Continuous Size	25
Attention Head Size	1
Learning Rate	0.0119*
Output Size	100
Loss	Quantile Loss
Quantiles	{0.00, … ,0.99}
Reduce on Plateau Patience	4

*rounded

3.2 Predicting Service Occupancy

At the beginning of each operating day, occupancy predictions are generated using the TFT Model for each service for the full current- and next-operating days using inference data derived from static timetable data. In order to make more accurate predictions as network conditions change throughout the day, the inference data is updated with real-time network information ingested from the PTV Timetable API (PTV, V3).

The PTV API provides direct access to Public Transport Victoria's public transport timetable data. The API can be queried to return scheduled timetable, route and stop data for all public transport modes. The API also gives access to real-time timetable updates for public transport services. Most important in this context is the ability to access real time metropolitan train timetable updates.

The data returned by the PTV API indicates whether a particular service has been cancelled, its expected arrival time changed, or it has been otherwise disrupted. This information is used to update the inference data, specifically the time varying known reals *Headway, HeadwayLine, SinTime, CosTime,* and *Disrupted.* The PTV API is called every 30 seconds and written to a database table using an update query.

Inference is conducted continuously in batches, where each batch contains all metropolitan train timetable information for the current operating day. Each batch of predictions takes between 10 and 15 minutes to generate and is picked up by the front-end website via an API. As soon as one batch completes, new timetable data is read, and a new inference job kicks off.

We make no attempt to predict the occurrence of unplanned disruptions, which are by their nature difficult to predict, so our ability to inform patrons of service occupancy in the presence of disruptions ahead of time is limited. However, when making predictions on disrupted services our model is able to capture the flow-on effects of disruptions and generate predictions

that include the impact that disruptions have across the network, and into the future timetabled services.

3.3 Quantile Cost Matrix

Although the aim of the model was to classify how busy a train was, we chose a regression model as our historical service load estimates are a continuous variable. We could have binned the historical estimates and used a traditional classification model, but this would effectively be discarding a lot of information (e.g., how close the 'busy' train was to being 'very busy'), which would reduce the accuracy of the model. We confirmed this intuition by creating a classification model and comparing the results.

Our original intent was to take the point-estimate from the model (which was an estimate for the median conditional on the independent variables) and simply use which-ever occupancy level that prediction fell into as our output that would appear in the application. We realised this approach had some serious shortcomings. The outputs consistently erred on the side of being conservative. The prediction class distribution looked different from the training data, being much less likely to predict 'busy' and virtually never predicting 'very busy'. These classes were rare in the training data, so it is not unexpected that this was the case, but we nonetheless wanted to have a more balanced class distribution. In a traditional classification model one can often adjust thresholds to deal with a class skew in the training data, but we had to find another approach given we were using a regression model.

A crude solution would have been to adjust the boundaries of each bin to artificially inflate or deflate the different categories. However, we also wanted to take into account the uncertainty associated with each prediction, e.g. if service A had a predicted service load of 200 (placing it as 'very quiet') and service B has one of 210 (also 'very quiet') it does not necessarily follow that service B had a greater chance of falling into a higher category. The TFT model might be very confident on its prediction about service B, in which case it would make sense to keep it as 'very quiet'. Conversely, there might be a risk of greatly underestimating service A due to high uncertainty, in which case we might want to bump up it up a category to prevent underestimation.

Our solution was to utilize quantile forecasting and generate predictions for all 100 percentiles. From this we could roughly reconstruct the probability density for each prediction, and then calculate the probability of each class for each prediction. For instance, if the predicted 90^{th} percentile equalled the cut-off between 'busy' and 'very busy' we could say that there was an approximately 10% chance of it being 'very busy'.

With the probabilities of each class, we could now make some adjustments so that the prediction class distribution was more to our liking. In particular we had two major goals:

1. The original predictions were overly conservative, leading to more under-estimation than over-estimation. We wanted the opposite to be the case, as commuters would be upset if the train was much busier than they anticipated.
2. We also wanted to hedge our bets and prefer the two central categories over the two more extreme categories, all else being equal. This means we wanted to punish instances where the predicted class was *very wrong*, i.e. more than one class away from the actual class, more so than when it was somewhat wrong, i.e. predicted an adjacent class.

We achieved this by creating a cost matrix. For each combination of predicted class and actual class, we have a cost. The cost where the predicted class = actual class is always zero, so the matrices have 0s along the diagonals. A cost matrix which simply chooses the class with maximum probability will be:

$$\mathrm{A} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

as each incorrect prediction is penalised equally.

Keeping in mind our goals of preferentially over-estimating and choosing the two central categories, we might choose a cost matrix that is skewed in one direction:

$$B = \begin{bmatrix} 0 & 2 & 4 & 8 \\ 1 & 0 & 2 & 4 \\ 2 & 1 & 0 & 2 \\ 4 & 2 & 1 & 0 \end{bmatrix}$$

where the asymmetry comes from us wanting to bias our predictions upwards, and the numbers get larger the further away from the diagonal to penalise predictions which are more egregiously incorrect.

Suppose for a single service at a station we have predicted a probability vector that looks like $\mathrm{x} = [0.5 \quad 0.4 \quad 0.1 \quad 0]^T$, which means we think half the time it will be 'very quiet', 40% of the time it will be not busy, etc. The simple approach of choosing the most likely class would be equivalent to

$$\mathrm{Ax} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.4 \\ 0.1 \\ 0 \end{bmatrix} = [0.5 \quad 0.6 \quad 0.9 \quad 1]$$

and then we choose the minimum cost prediction, which is 'very quiet' at 0.5.

Using the cost matrix, which is designed to achieve our goals, B, the same equation yields:

$$\mathrm{Bx} = \begin{bmatrix} 0 & 2 & 4 & 8 \\ 1 & 0 & 2 & 4 \\ 2 & 1 & 0 & 2 \\ 4 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.4 \\ 0.1 \\ 0 \end{bmatrix} = [1.2 \quad 0.7 \quad 1.4 \quad 2.9]$$

and the minimum cost prediction is now the second category, 'not busy' at 0.7. The cost matrix has moved our prediction up by a category, which is probably a good idea, not least because there's a 10% chance the true category is actually 'busy' and we really want to avoid predicting 'very quiet' when the true class is 'busy'.

The cost matrix can be adjusted to taste so as to achieve the desired class distribution and confusion matrix, or even adjusted over time, if necessary, including in cases where pandemic restrictions wreak havoc on transport patronage and class skew.

3.4 Preventing Model Drift

Each week the model undergoes a retraining procedure, where the current best model according to the evaluation metric is loaded and fine-tuned for several epochs, using the same early stopping mechanism described in 3.1 to decide when to end the retraining. Afterwards, the new model's evaluation score is compared to the previous model's score. If the new model has produced a better score on the evaluation metric, it is then pushed to production, and the previous model version is archived. Otherwise, the new model is archived instead. The evaluation criterion is defined as follows:

$$\sqrt{MAE} \times SMAPE$$

Where MAE is the mean absolute error, and SMAPE the symmetrical mean absolute percentage error. Both metrics are calculated using the newly added training data only. MAE is dependent on the actual mean departure boardings, which may vary week-to-week. Therefore, we take the square-root and multiply this by the percentage error term to reduce the variance.

4. Results

Key indicators of model accuracy are summarised in *Table 3*. Instead of having a single TFT model trained and fitted across all metropolitan train services, we have divided the rail network into four sub-networks, separated based on where each train line enters the central city loop. Melbourne's rail network is circular, with all train lines eventually merging in this central loop. Each train line grouping is defined by which of the four city loop tunnels that particular line usually enters the city loop as follows:

Burnley tunnel group: Alamein, Belgrave, Glen Waverley and Lilydale lines
Caulfield tunnel group: Cranbourne, Frankston, Pakenham and Sandringham lines
Clifton Hill tunnel group: Hurstbridge and Mernda lines
Northern tunnel group: Craigieburn, Sunbury, Upfield, Werribee and Williamstown lines

The line groupings are also highlighted visually in *Appendix B*. This has advantages in allowing us to compare model performance for separate sections of the network and retrain individual models for better prediction accuracy. However, the main purpose of this is to allow us to fit longer encoder sequences into each training batch, while staying within the memory limits of our GPU hardware.

Table 3: Training Results for all 4 TFT models

	Burnley	Caulfield	Clifton Hill	Northern
Parameters	342,000	**343,000**	342,000	**343,000**
Epochs	**76**	94	78	83
Loss	7.66	6.66	5.70	**5.67**
MAE	21.21	18.33	15.88	**15.54**
SMAPE	0.41	0.51	**0.38**	0.54

While the TFT models predict train service departure loads at each stop as a continuous variable, the predictions seen by end users are transformed crowding categories, which are generated by the quantile cost matrix as explained in section 3.3. Therefore, the key metric we use to measure model accuracy is adjacent category accuracy, where we label a prediction as "correct" if it was at most one category away from the actual observed crowding level. Our intention is not necessarily to predict the exact level of crowding as often as possible; instead, we want to avoid under-estimating the actual level of crowding primarily.

Figure 2 shows the category accuracy matrix. It can be observed that the true category distribution is heavily skewed towards one end, with roughly 90% of all station departures being categorised as 'very quiet', while only 0.16% of all services are 'very busy'. Predictions from all four TFT models combined were exactly accurate 86.2% of the time and had an adjacent category accuracy of 99.1%. We can see that the biggest error is when the model predicted 'busy' but the actual category was 'very quiet', which occurred 0.55% of the time. This error in overestimation could be addressed by not applying our cost matrix to the predictions, but as our intention was to minimize under-prediction, we instead accept this trade-off.

Figure 2: Category accuracy matrix for predictions from May 2022. Adjacent category accuracy is shown in the green areas, while errors are shown in red and pink regions.

	Predicted Label			
True Label	1 Very Quiet	2 Not Busy	3 Busy	4 Very Busy
1 Very Quiet	81.073%	8.649%	0.549%	0.004%
2 Not Busy	2.652%	4.828%	1.138%	0.021%
3 Busy	0.162%	0.490%	0.263%	0.013%
4 Very Busy	0.050%	0.068%	0.038%	0.004%

The 'very busy' category is hardest to predict – only one in four 'very busy' services was correctly classified, and the majority of these correct predictions are in the adjacent category 'busy'. One key limitation of the model, which is discussed in more detail in section 5 is the inability to accurately estimate crowding due to special events such as sporting events or concerts. This is highlighted in *Figure 3*, where the network models are unable to predict the 'very busy' services occurring between 10 pm and midnight. *Figure 4* shows results for a week in May 2022 without any major events. Here the models were able to predict within one category for 'very busy' services 63.7% of the time.

Figure 3: Distribution of actual (left) and predicted (right) crowding category at each hour of the day.

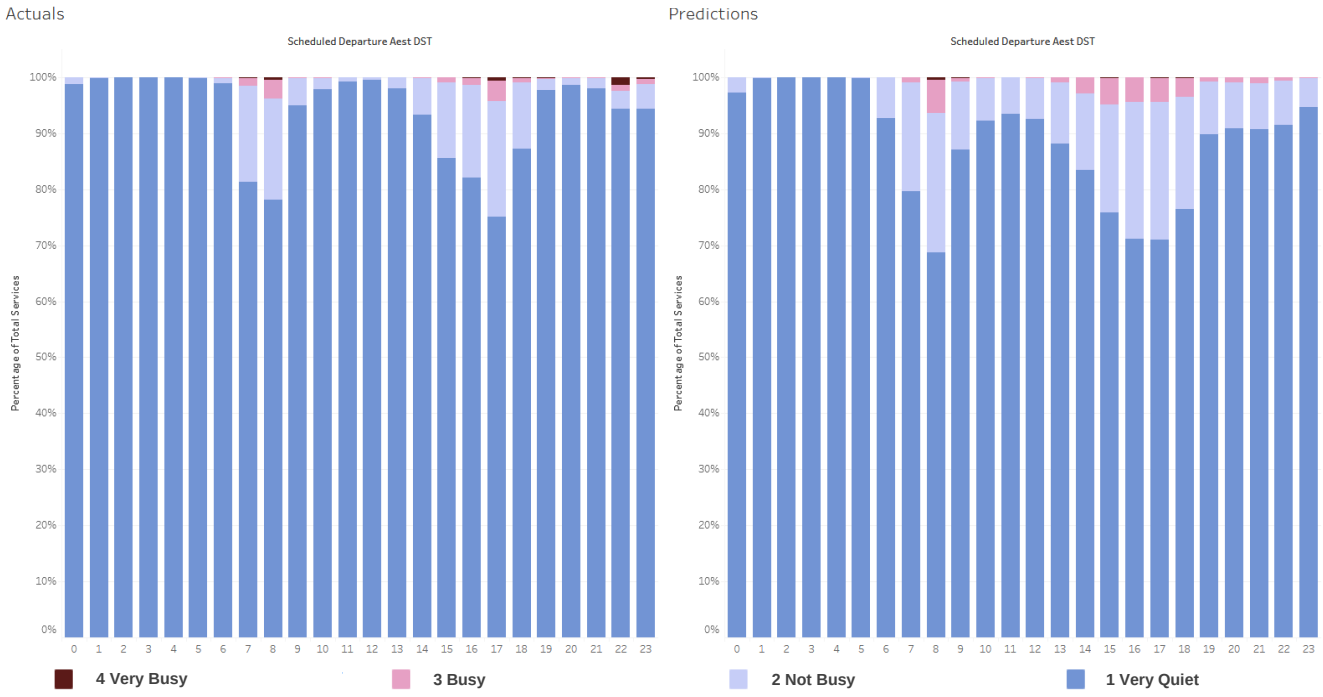
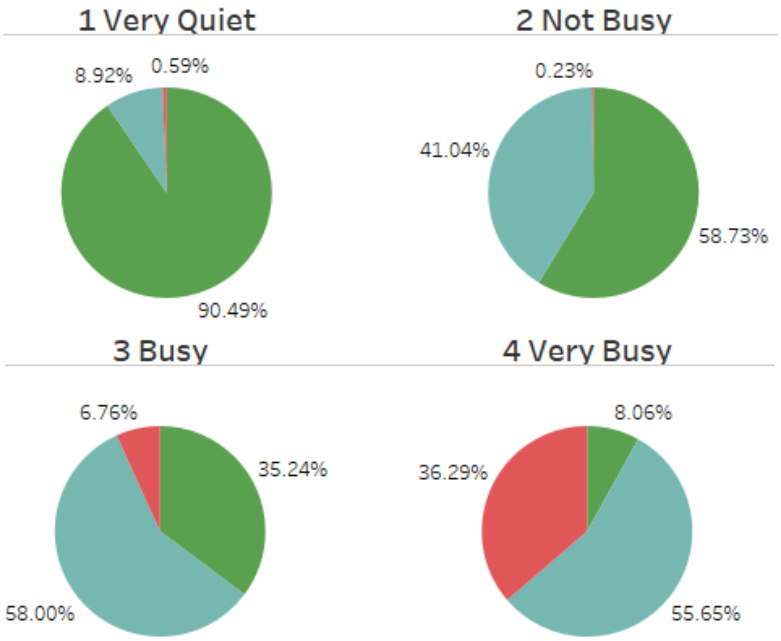


Figure 4: Category level accuracy from days in May 2022 without major disruptions or special events. Here green = prediction matches actual, light blue = predicted category is one below or above the actual category (i.e. adjacent) and red = the prediction is more than one category away from the actual.



5. Limitations

The main limitations of our modelling approach are caused by difficulties accessing data. Firstly, while the model does ingest real-time timetable data, due to hardware limitations it has no access to real-time ticketing or patronage data and ingests ticketing data only the day after it occurs. Ideally, a solution would access ticketing or automatic passenger counting data in near real time to detect unusual spikes or troughs in patronage and modify predictions accordingly. This kind of data would enable a spike in trips at the beginning of a service to be communicated to passengers further down the line (or across the network) in the form of updated predictions. Currently, the solution is unable to react to real-time fluctuations in passenger loads.

Further, the introduction of more datasets to the model would enable greater accuracy in predicting crowding levels as a result of special events, or unplanned disruptions. For example, large spikes in patronage around events such as football games are relatively predictable if data can be provided to the model regarding the upcoming schedule of such events. In particular, we have observed that when an AFL game is played at the Melbourne Cricket Ground (MCG) at night-time on a weekday, the increased patronage displays a predictable pattern if the game is known about, but in the absence of this information the model cannot generate accurate predictions. This kind of 'special events' dataset does exist (and importantly includes expected attendance), and work is being done to incorporate it into future iterations of the model.

Similarly unplanned disruptions such as accidents or other incidents that cause major delays or cancellations are not directly labelled in our training data. Instead, the model must infer when these occur based on changes in headways between train services - far from ideal. Unexpected incidents are extremely hard to anticipate, but with knowledge of when these events have happened in the past, the TFT would have the ability to learn how to react in such situations, which would significantly improve the time it takes for crowding estimates to update and adjust in real-time.

6. Applications

The TFT architecture is flexible and has been applied to several timeseries forecasting problems in (Lim et al. 2019), including electricity prices, retail sales and traffic volumes. However, these are all more or less toy examples and do not bear any considerations for scaling to large datasets. Our work has extended this by showing how to successfully implement a production ready TFT model that can generate and update predictions in real-time.

Our implementation could easily be transferred to other public transport networks, the only requirements are input data on passenger boardings and train loads, as well as timetables of scheduled services. For real-time inference, the only additional requirement is that timetable information is updated with delays and cancellations at a reasonable frequency. If historical data on special events and unplanned disruptions was also available, the majority of the current limitations could also be addressed. Additional real-time data feeds such as ticketing transactions or passenger counts would help improve the accuracy and speed at which predictions could be adjusted based on actual network usage.

Another key application is in short term rail planning. The model is able to analyse rail timetable schedules and train frequencies to determine the possibility of extreme crowding. The model is sensitive to both the headway of each individual train route, and also the headway between all services travelling in the same direction at a given stop. This means the model is able to deal with complex timetable changes, including shifting services from one line to another. Because the model was trained on data during Victoria's Covid-19 recovery, it is also able to be used for scenario planning, i.e. modelling expected crowding when lifting pandemic lockdown restrictions.

Citations

Bryan Lim, Sercan O. Arik, Nicolas Loeff &Tomas Pfister (2019) Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting, arXiv, DOI: 10.48550/ARXIV.1912.09363

Debaditya Chakraborty & Hazem Elzarka (2019) Advanced machine learning techniques for building performance simulation: a comparative analysis, Journal of Building Performance Simulation, 12:2, 193-207, DOI: 10.1080/19401493.2018.1498538

Department of Victoria (2021) Ridespace, Ridespace Website: https://ridespace.coronavirus.vic.gov.au/

Jan Beitner (2020) Pytorch Forecasting, GitHub Repository: https://github.com/jdb78/pytorch-forecasting

PTV API (V3) Licensed from Public Transport Victoria under a Creative Commons Attribution 4.0 International Licence. https://www.ptv.vic.gov.au/footer/data-and-reporting/datasets/ptv-timetable-api/

Public Transport Victoria (2018) Victorian Train Network Map, PTV Website: https://www.ptv.vic.gov.au/assets/PTV-default-site/Maps-and-Timetables-PDFs/Maps/Network-maps/0c96079d1f/Victorian-train-network-map.pdf

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama (2019) Optuna: A Next-generation Hyperparameter Optimization Framework, in KDD, DOI: 10.1145/3292500.3330701

Appendix A: Train Service Usage Model Algorithm

Statistical model that combines total patronage (station entries and exits) with the actual service timetable departure and arrival times, to assign users to the most likely train service (and any transfers if applicable) for their journey. After assignment, the model calculates boardings, alightings, arrival loads, departure loads and transits for every train service at every stop (station).

The model generates 3 types of trips:

- Paired trips (p)
    - Passengers touch on and off.
- Unpaired trips (n)
    - Passengers did not touch off.
- Ghost trips (g)
    - Passengers did not touch on at all.

We know ghost trips exist due to our Touch On Rate Survey results that infer patronage. The model creates them by splitting the boost factor from each trip into two components:

- A single trip, representing the passenger who touched on
- The remainder of the boost factor (ie. boost factor - 1)

For example, a transaction with a boost factor of 1.25 would be split into a single trip with a weight of 1.0, and a ghost trip of weight 0.25.

Passengers travelling to the CBD are much more likely to touch-on as they know they are likely to have to touch-off at the gated CBD station. To reflect this, in the imputation model that infers missing touch-offs, ghost trips are prevented from going to gated stations. Unpaired trips are still permitted to have their touch-off imputed to be a gated station.

The imputation model is needed to create synthetic touch-offs for ghost and unpaired trips. It works by creating a 'sampling frame' of all paired trips, and grouping them by:

- Touch On Station
- Touch Off Station
- Day Type
- Touch On Time Period (0-5am, 5-10am, 10-3pm, 3-8pm, 8pm-midnight)

The model is usually run in weekly batches, so the sampling frame will represent a week's worth of sampling data. The average journey time is calculated for each group, and each unpaired/ghost transaction is randomly assigned a touch off station within the same grouping. Each unpaired/ghost touch-on is randomly assigned a paired trip from the sampling frame within the same group (i.e. same touch on station, same day type, and same touch on period). This assigns a touch-off station to the unpaired/ghost touch-on. While the vast majority of trips are imputed using this method, the model then iterates through this again for any trip that was not successfully imputed, but without the condition on the time period to try and match all remaining trips. Once a touch-off has been assigned to each unpaired/ghost touch-on, the average journey time is added to the touch-on time to impute the touch-off time.

1 Appendix B: Melbourne Train Network Line Groupings (PTV, 2018)