Train ETA Prediction for Rumo S.A.

Aidan Claffey
adc501
aidan.claffey@nyu.edu

Dan Turkel
dgt238
dan.turkel@nyu.edu

Kevin Wilson
ksw366
kevin.wilson@nyu.edu

NYU Center for Data Science

1 Introduction

Rumo S.A., a subsidiary of Cosan Group, is the largest freight railway operator in South America. The company operates an extensive logistics network in Brazil consisting of 13,500 km of railway; 1,200 locomotives; and 33,000 wagons; and it serves customers primarily in the agricultural commodities and mining sectors.

The company's network is largely single-tracked, meaning that crossing yards are required to enable simultaneous outbound and inbound train traffic. Rumo operates at very high capacity (over 300 simultaneous active trips at all times) with competition, customer demands, and profitability goals driving higher capacity targets. The coordination of several hundred daily arrivals and departures is central to the company's operations, and thus predicting each train's estimated time of arrival (ETA) accurately is critically important.

Rumo's current ETA estimations are built on a discrete event simulation with uneven performance across the network's northern and southern regions. Our task was to develop a new, data-driven approach to improve ETA prediction accuracy.

Our approach was to transform historical train traffic and train composition datasets such that we can create a regression model to predict estimated time of arrival for new data. We implemented several models, the most successful of which were a multi-layer perceptron and histogram-based gradient boosted tree model. Our extensive feature engineering, creative approaches to reformulating the task, and substantial modeling and tuning led to significantly higher prediction accuracies than Rumo's existing methodology. In the process, we learned many valuable technical skills and practical lessons.

2 Related Work

While the general problem of ETA prediction is well-studied, existing research and solutions are largely domain-specific. More research has been completed on public transit related estimated arrival time solutions [2] [7] than on railway freight. This could be due to more publicly available data on public transit than on freight, which is generally owned and operated by private sector entities. One of the few studies we found on railway freight ETA [1] is limited in the scope of its task and its limited geographical region.

European freight groups in partnership with the European Union have organized a research project called ELETA,[1] or Electronic Exchange of ETA Information, that aims to use modern machine learning methods to improve ETA times for railway freight across Europe. Additionally, the ELETA project notes that clean, transparent, and standardized data collection is part of their end goal. However, it is unclear when the research will be complete, or whether the findings will be made publicly available.

Although freight trains are used around the world, this project is firm-specific and difficult to generalize to a broader context. There is no standardized data collection or data presentation method

[1]Artificial Intelligence Improves Estimated Time Of Arrival - Railfreight.com

used in rail freight logistics, so modeling efforts often need to be adapted to the available data and metrics of interest. While there may be movement in that direction in the future, the scope of this task is intended to specifically improve Rumo's logistics.

3 Problem Definition

3.1 Task

The task was to develop a machine learning approach to estimating time of arrival (ETA) for all of Rumo's trains that achieves region- and time-based accuracy targets: 90% and 65% accurate in the South and North regions, respectively. An accurate prediction is defined as having an error of less than 1 hour for predictions between 6 and 12 hours of the destination and less than 2 hours for predictions between 12 and 24 hours of the destination. These were ambitious targets, be we hoped to develop an effective, high performing, data-driven model to be deployed on new train traffic data. Our methodology, including our approach to formulating this task, are discussed below.

The main deliverable requested by the company was the Python code implementing our model, with descriptions of any inputs and outputs, engineered features, and an explanation of our approach and methodology.

3.2 Data

Rumo records and maintains a multitude of sensing data, train traffic and composition time series data, and semi-static railway infrastructure information that are relevant to this task. The company provided two main datasets and several supplementary files. Column names for all files were in Portuguese; herein we refer to English translations for clarity.

3.2.1 Train Traffic

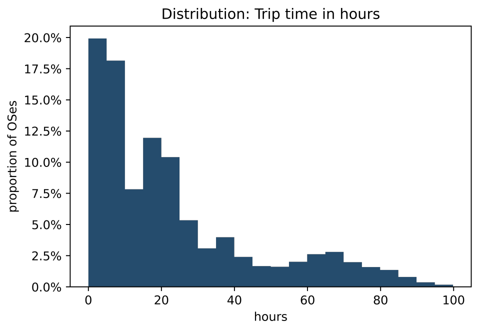
The first and primary dataset is a granular, time series dataset called Train Traffic. It contains one row for every Section Block every train enters and exits, and records the entry and exit time (`ts_enter` and `ts_exit`, respectively) and total time in seconds each train spends in each Section Block, where a Section Block is the smallest unit of track under consideration, about 1 to 40 kilometers long. Rows are labeled with `OS` (Order of Service, a unique trip indicator, which allows for tracking a train through its journey in the network) and `train_prefix` (3-digit code that captures direction and route). The total time in Section Block is further disaggregated into five categories at the (`OS`, Section Block) level:

- `time_moving_in_section` (`CONDUCAO`): Time in seconds in which the train was moving through the Section Block
- `time_in_yard` (`PERMANECIA`): Time in seconds spent by the train in maneuver yard
- `time_waiting_crossing` (`CRUZAMENTO`): Time in seconds spent by the train waiting the other train to go through the Sectiotn Block
- `time_for_anomaly` (`ANOMALIA`): Time in seconds in which the train did not move due to any anomaly in the railway, or safety device alarm or mechanical problem
- `time_for_anomaly_ahead` (`PROPAGACAO`): Time in seconds in which the train did not move due to an anomaly ahead that made other train stop and caused this train to stop

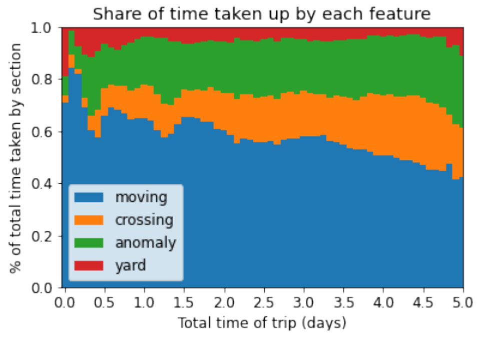
Summing these fields across a given row yields the total time that train was in that Section Block, and that sum is equivalent to (`ts_exit - ts_enter`). An `OS` travels through about 100 Section Blocks on average, and there are several hundred unique `OS`es operating on a given day, with approximately 10,000 Train Traffic entries per day across all `OS`es.

At first we only had access to data for October 2019, which we used for exploratory analysis and preprocessing. Ultimately we were given access to data covering September 2019 to September 2020, with only March 2020 missing due to an internal issue at Rumo. This dataset included 3,531,276 unique entries.

There are 1,366 unique Section Blocks visited by trains in the dataset, with each Section Block having the capacity for at most 1 train. Rumo's network is encoded hierarchically, and each Section



(a) Trip time (origin to destination) varied, with a mean of 23.6 hours. Excludes 0.3% of trips > 100 hours.



(b) As total time increases, share of time spent waiting for crossing or anomaly increases

Figure 1: Selected exploratory data analysis

Block is part of a designation of track called a Local, which may contain one or more Section Blocks. There are 982 unique Locals, with 633 (64.5%) containing just 1 Section Block; Locals containing 2 or more Section Blocks are either crossing points or maneuver yards. We determined this level of specificity was best for making predictions.

Locals comprise 72 sub-stretches, which in turn comprise 47 stretches and 13 corridors. The two broader regions, North and South, contain 4 and 9 corridors, respectively, and together constitute the entire network. See Figure 4 for a visual representation of this hierarchy.

3.2.2 Train Information

Rumo also provided a time series dataset with more detailed train-specific characteristics, but at coarser time intervals than Train Traffic. The Train Information dataset includes one row for each train "composition" for each trip (with a single `OS` consisting of between 2 and 3 distinct compositions, on average), with columns encoding the different products carried by the train, the number of locomotives, the number of wagons, train length, train weight in tons, and various other fields. Rows are additionally labeled with train prefix, type of wagon and locomotive model, and origin and destination Section Blocks.

Rumo provided a series of Microsoft Excel files covering January 2019 to October 2020, inclusive. This dataset included 200,822 unique entries.

3.2.3 Supplementary datasets

In addition to the above, Rumo provided several supplementary files from which we were able to derive useful additional features. These included information on the length of each Section Block in kilometers, the capacity (in number of trains) of maneuver yards, and the set of rules describing the signaling and advancement procedure for trains actively moving through the network. We also had data on predictions made from the currently used simulation, and the resulting errors, for the month of September 2020.

3.2.4 Preprocessing and transformation

The size and complexity of the raw data called for significant merging and transformation to effectively incorporate all available information in a condition appropriate for supervised machine learning. This included substantial data cleaning, removing duplicates modifying incorrectly encoded outliers, translating some elements from Portuguese to English, and merging Train Traffic and Train Information data.

3.2.5 Feature engineering

We engineered and computed a large number of additional features to extract network-realted information from the data from the time series data. Each of the below were computed at the (`OS`, `Section Block`) level: that is, for each row of the processed and merged Train Traffic dataset. The full engeineerd feature set is enumerated in Table 2.

Trains Ahead: 16 features characterizing the behavior, direction, and location of trains in the yet-to-be-visited Locals of a given `OS` at time `ts_exit`. These features encoded information that we expected to be correlated with future waiting time. For example, having a higher than usual number of trains ahead moving in the opposite direction could indicate that a train will be more likely to need to wait at a crossing point or stop in a maneuver yard.

Crossings Ahead: 8 features characterizing trip-to-date performance and expectations for future number of crossings (defined as *count* of non-zero `time_waiting_crossing` for an `OS`'s yet-to-be-visited Locals) and total future time waiting crossing (defined as *sum* of `time_waiting_crossing` for an `OS`'s yet-to-be-visited Locals). Performance and expectation comparisons were computed based on rolling means of trips with the same `train_prefix` over the last 14 and 60 days.

Yards Ahead: 8 features analogous to Crossings Ahead but for Locals defined as maneuver yards and for the time field `time_in_yard`.

Stretch Congestion: 6 features characterizing the congestion of the current and next-to-be-visited stretch, computed as both *absolute* congestion (number of trains "in" current/next stretch) and *relative* congestion (compared to historical means for trips in the last 14 and 60 days).

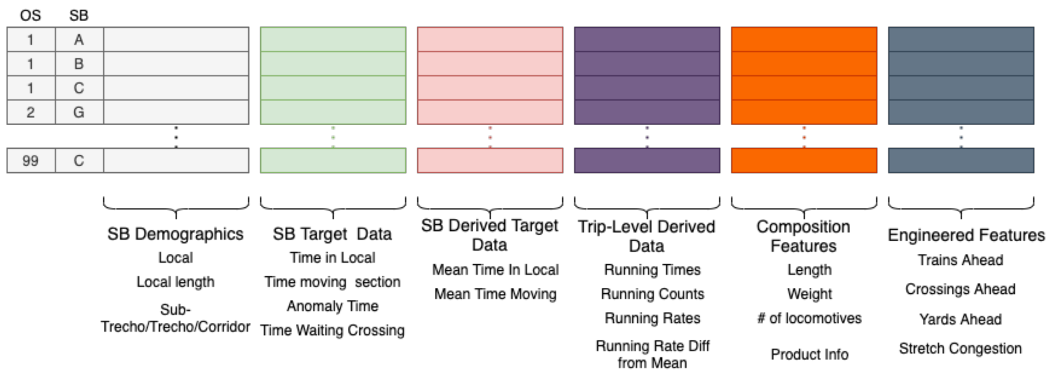


Figure 2: State of merged data after preprocessing and feature engineering.

3.3 Methodology

The task of ETA prediction can be constructed in a many different ways. For example, given a starting and ending locations, one could predict the aggregate time of the total journey. Alternatively, given a train's current location and all subsequent Section Blocks it will pass through, one could predict the length of time the train will take in *each* Section Block. Or, given the entire state of the train network at each step, one could graph the network and use graph-based methods for prediction. For our purposes, the first of these was too simplistic; the last demanded more data and resources than we had access to. Thus, we formulated the task to predict the total time each train spends in each following Section Block.

3.3.1 Target variable choices

As noted in Section 3.2.1, the time spent in each Section Block can be decomposed into five features. The `time_for_anomaly_ahead` feature is so infrequently non-zero that we merge all of its instances with `time_for_anomaly`. We consider each of these four features as target variables.

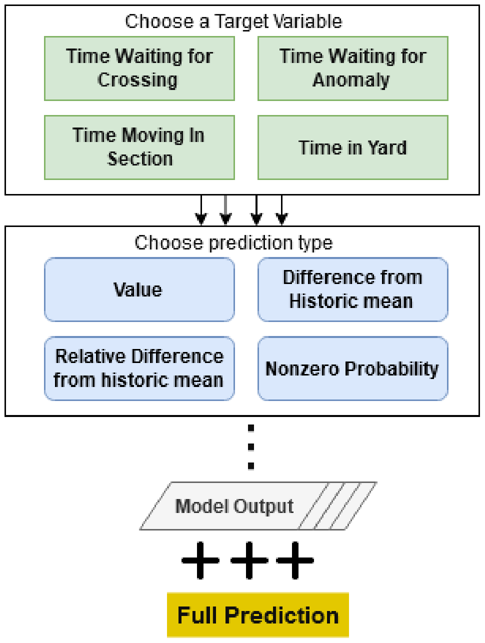
As a technical note, although the historical data is at the Section Block level, we are *actually* predicting at the Local level, where a Local typically contains one or two Section Blocks. For Locals that have more than one Section Block, we do not know *a priori* which Section Block in a Local the train will go through.

Additionally, each target variable can be represented in a few different ways. If the historical means of each target variable for each Local are stable and representative of the value itself, it may be easier to predict the difference from the historical mean, or even the relative difference from the historical mean. The *relative difference* prediction is even more useful when considering the fact that Section Block lengths vary widely (from 0.5km to greater than 10km) and naturally time in Section Block scales roughly linearly with this length. Thus, trying to predict the value or the difference from mean for both of these Section Blocks will be difficult. Using *relative difference* from mean will put the targets for these two Section Blocks on the same scale.

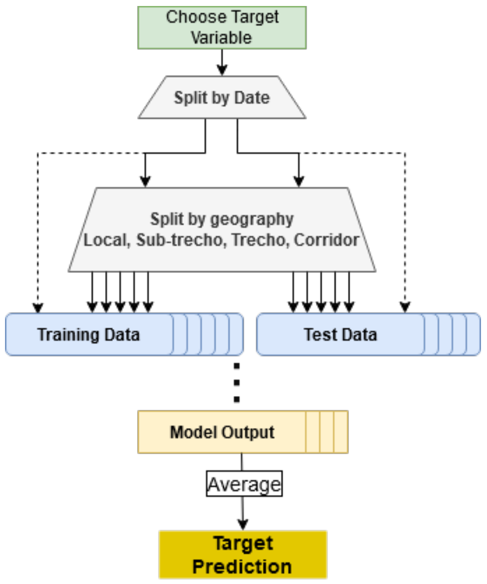
We also figure that predicting the *probability* of a nonzero value for a target variable, and then either assigning or predicting a value based on a positive prediction, may work better for target variables with many zero values. This is true for `time_for_anomaly` (11% non-zero) and `time_waiting_crossing` (7% non-zero). Figure 3a represents this architecture.

3.3.2 Model training choices

After we have selected the target variable we wish to train, we developed an ensemble architecture to that takes advantage of the geographic hierarchy of the data. We decided that training a model only with data from the same Local will help predict small differences between train trips going through the same Local, whereas training on larger geographic groupings will allow for broader themes to be detected. Modeling at each geographic granularity will provide different results, so we enable the ability to pick one level of granularity or to model all of them and average the results. Figure 3b represents this architecture.



(a) To most accurately predict total time in Local, we individually model each target variable.



(b) Each model can be trained using only data from specific geographic representations, or it can be trained on the full dataset (dotted line)

Figure 3: Model choices and architecture

3.4 Model

3.4.1 Historical Mean Baseline

As both a first-pass attempt at the task and a baseline to evaluate more complex models, we built a model which makes predictions based on historical means. Rumo noted that for a full *trip*, from start to finish, using the average historical time taken for that trip as a prediction would not perform well. Indeed, trips can have tremendous variance in travel time.

However, the variance in trip time arises from a small piece of the larger trip. Dividing the trip into its constituent Locals and dividing each Local into the time spent moving, waiting at a crossing, waiting in a yard, or waiting due to an anomaly (or anomaly ahead) allows us to utilize much more granular historical means. In fact, time moving in each Local has extremely low variance and can be predicted reliably from historical averages alone (90% of all `time_moving_in_section` entries are within 5 minutes of their historic local mean).

We formulate our baseline as slightly flexible in construction, to provide a better point of comparison with our configurable model architecture described in Section 3.3.2. Models which predict only a single category of time (e.g. time waiting at a crossing) can be evaluated against baseline predictions made solely for that category (historical mean of time waiting at crossing per Local). Models which make predictions for each category of time, however, can be evaluated against baseline predictions made using the sum of each category's historical mean.

The performance of the historical mean model is a function of how much data is included in the calculation and how recent it is, compared to the test data. We experimented with using 2 weeks, one month, 6 weeks, and 3 months of historical data to set the baselines and found that one month led to the strongest baseline for naive trip length estimation. However, we tested this only for a single point in time, and it is possible that the appropriate amount of data may change at various points in the year due to seasonal effects. Our model architecture allows for the historical baseline period to be set separately from the training data.

3.4.2 Multi-Layer Perceptron

Our first noteworthy model used the most simple form of a feed-forward nerual net, the multi-layer perceptron (MLP), as its predictor. The general architecture of the MLP had two hidden linear layers with a ReLU non-linearity after the output of each hidden layer. We experimented with dropout, but it did not add predictive power to the model so we ended up with a fully connected, 4 layer network, including the input and output layers.

The MLP was used specifically to predict the probability of a non-zero value for `time_for_anomaly` and `time_waiting_crossing`. Then additional MLPs were used to predict the values of only the positive predicted classes for those target values at each level of geographical granularity. The historical means were used for the `time_moving_in_section` and `time_in_yard`

For the classification models, we used a weighted Binary Cross Entropy loss to deal with class imbalance, and for the regression models we used a Mean Squared Error loss. The hidden size for each linear layer was 1024, and the learning rate scheduler we used was Adam with a learning rate of 0.001. The model was implemented in PyTorch [5].

3.4.3 Histogram-based Gradient Boosted Regression Trees

Our second noteworthy model used histogram-based gradient boosted regression trees, as implemented in Scikit Learn [6].

Gradient boosted trees are an ensemble method similar to random forests, but rather than learning many trees and aggregating their results, we learn many trees *in sequence*, where each tree attempts to fit the residuals of the prior tree. This sequential nature can lead to poor performance. To make matters worse, growing decision trees with many continuous features can already be slow as traditionally every possible split of a feature is checked to see if it is the optimal split for a node in the tree.

Popular gradient boosting libraries LightGBM [4] and XGBoost [3] have achieved much faster training performance through use of histogram-based tree growing, where continuous features are discretized into bins containing roughly equal numbers of data points. Growing a tree now requires only evaluating the splits between each *bin* of the feature (a much smaller number than the splits between each value), and there is no longer any sorting required.

Our histogram-based gradient boosted regression tree model learns four boosted regression trees for each Local in the network: one for relative deviation from mean time moving and one each for the value of the time waiting for a crossing, time in a yard, or time due to an anomaly. Since this comes out to several thousand individual boosted tree models, the performance gains of the histogram-based implementation add up to make this low level of granularity feasible to train.

4 Experimental Evaluation

For our multi-layer perceptron and histogram-based gradient boosting trees, we trained on data from September 20–26, 2020, and evaluated on data from September 27–30th, 2020. Our historical means were calculated on September 1–26th, 2020.

4.1 Evaluation Methodology

As mentioned in Section 3.1, Rumo's goal for the end of the project was to reach an accuracy of 90% for the South region and 65% for the North. The reason for this discrepancy is that the North region has higher traffic congestion and has a larger proportion of single-tracked railway compared to the South. While these accuracy goals are quite ambitious, we hoped to do improve on Rumo's existing model's accuracy rates, which were approximately 37% and 55% for the North and South, respectively, for September 2020.

Since our predictions are made at the Local level, we need to aggregate them into trip-level predictions. We do this by considering all sub-trips for a single `OS` that take between 6-24 hours as separate trips. A sub-trip is defined as any collection of contiguous Locals in a trip that maintain the same order as in the original trip. Then within each sub-trip, we sum up all Local prediction to get the trip-level prediction. A sub-trip prediction is considered accurate if it's predicted time is within a certain threshold of its true time (1 hour for sub-trips of 6-12 hours; 2 hours for sub-trips of 12-24 hours).

4.2 Results and Discussion

Table 1 shows the model results for our 3 models and the discrete event simulation model (labeled Simulation).[2] The Multi-Layer Perceptron performed the best in the North region, with accuracy well above the simulation at both thresholds for each trip length.

All models performed better in the South than in the North. Somewhat surprisingly, however, the best model for the South was our Historical Mean Baseline. We did not achieve the accuracy targets Rumo set, but we were pleased to have improved on the current implementation in both regions and at both trip scales. The dramatic improvements in the 6–12 hour trips are particularly promising, with even our weaker model gaining 12 percentage points on the Simulation in the South.

The data reveals how Mean Absolute Error is not sufficient to tell the story of each model's performance, given the long-tailed distributions of the trip lengths. For example, the baseline and gradient boosting model both have MAE of 2.2 hours on 6–12 hour trips in the North, but gradient boosting is more accurate within the 1-hour threshold by over 6 percentage points. Future exploration could include training models which optimize for trip-level prediction accuracy directly, rather than Local-level error.

Table 1: Model accuracy for the discrete model simulation, the historical means baseline, the multi-layer perceptron (MLP), and the gradient boosting (GB) model. In **bold** are the best results for each metric in each section.

	North 6-12hr		**South 6-12hr**	
	MAE	**Acc ±1hr**	**MAE**	**Acc ±1hr**
Simulation	-	22.5%	-	36.8%
Historical Means	2.2hr	26.9%	**1.0hr**	**58.5%**
GB Model	2.2hr	33.0%	1.3hr	48.4%
MLP Model	**2.0hr**	**38.1%**	**1.0hr**	55.7%
	North 12-24hr		**South 12-24hr**	
	MAE	**Acc ±2hr**	**MAE**	**Acc ±2hr**
Simulation	-	39.9%	-	64.4%
Historical Means	3.6hr	31.1%	**1.4hr**	**77.3%**
GB Model	3.2hr	38.7%	1.7hr	67.5%
MLP Model	**3.1hr**	**44.4%**	1.7hr	70.1%

[2]Simulation errors are calculated from a set of simulations run for all of September 2020, and is only for full trips 6-12hrs (not subtrips), so it is not a perfect apples-to-apples comparison with our models, which were evaluated on all 6–12 and 12–24 hour subtrips in the September 27–30 test set.

We were surprised to see how much our models' performance varied across the regions, even given our understanding of the regional differences. In the case of the current Simulation approach, the rules were designed according to dynamics of the South region, which consists primarily of single-track routes, and when it was applied to the newer northern region, Rumo found it didn't perform as well. But since our methods learn models for each individual Local, there is no obvious reason why the contrast between regions would be so stark. A clue may lie in the fact that performance also varied greatly across *corridors*, another geographical unit of the network, suggesting that different sections of the network may experience different dynamics that our feature set does not capture.

Table 1 shows only results on the test set. It should be noted that results on the training data were better (dramatically so in the case of the gradient boosting model, which achieved 84% accuracy ± 1 hour in the training set). While we were able to reduce the amount of overfitting with some hyperparameter optimization and experimentation, it was clear that the models were still not optimally tuned since they were not generalizing well to out-of-sample data. However, the capacity for the models to overfit is promising, especially considering that the training data was larger than the test data: the models are capable of learning feature relationships relevant to the high-variance target variable. Our training time on the relatively small training set was relatively large, but model generalization could potentially be improved by increasing the size of the training data.

Some of our earlier models which used random forests for regression allowed us to see which features the model found most informative. A train's gross tonnage, which could impact its speed, was found to be the most important feature. Others included the length of the train and the number of empty and full wagons (very long trains cannot fit in crossing yards and thus get priority to cross when competing with a shorter train), several measures of the number of upcoming train yards, and how far into the trip the train was.

MLPs do not offer a straightforward notion of feature importance, and unfortunately feature importance is not yet available in the Scikit Learn implementation of histogram-based gradient boosting trees. The latter is a missed opportunity for us, as we do not get visibility into which features informed one of the strongest-performing models we tested.

4.3 Error Analysis

We performed an in-depth error analysis on our final gradient boosting model. In addition to the high variance across regions and corridors in the network, we found several surprising trends. Within the 6–12 hour trips, we expected that longer trips would have greater prediction error, but we found no such correlation. Similarly, there was no correlation between prediction error and number of Locals visited in a 6–12 hour trip. One explanation is the tendency for overestimation in some Locals to be canceled out by underestimation in others.

The distribution of trip estimation errors was centered at zero, with most of its mass within -8 to 8 hours, but several outliers show our model overestimated some trips by over 30 hours. We suspect that these errors are due to overfitting in our anomaly time prediction. We experimented with building only one model for anomalies (rather than one model per local) to reduce this issue but did not see a great improvement. Given more time, we would more closely examine the error distributions of each individual time category: early tests showed that prediction errors for time moving were very low, but all other times tended to have high error and high variance.

The distribution of baseline trip errors was slightly different from the model errors, with its center lying at about an hour of overestimation rather than zero. The mass was again distributed between -8 to 8 hours, but this time with a heavier tail toward *underestimation*, which is counterintuitive and warrants further exploration. As with the model errors, in the future we would like to break down the distributions into the separate components of trip time.

4.4 Implementation

Rumo emphasized that while the end results were important, they also felt that the intermediate steps, such as feature engineering and model iteration, were an important deliverable. Additionally, Rumo planned to use our findings to continue research on improving their ETA prediction system, not necessarily as *the* solution for their ETA prediction system. So, we focused some of our time on building a modular end-to-end prediction system that can preprocess the data, do feature engineering, apply a variety of prediction models on any subset of the data, and evaluate the results.

4.4.1 Preprocessing

Our `RumoPreprocessor` module takes raw data given to us by Rumo and prepares it so that the we can use it for modeling. This includes preprocessing described in Section 3.2 as well as adjustments to `python` data types to keep data size small.

The `RumoTargetTransformer` allows us to join our historical mean times to our training and test data, as described in Section 3.4.1. We can `fit` the transformer on historical data to calculate the average times (moving, waiting, in yard, etc.) for each Local—optionally calculating a separate average per Local for inbound versus outbound trains. Then we pass our training and test data to the `transform` method, which joins our historical averages to the data by Local (and optionally direction). Our output now has actual times and historical times side-by-side, allowing us to easily derive the difference from mean or relative difference from mean targets as described in Section 3.3.1.

The `RumoTrainsAhead` module takes as input the preprocessed and merged train traffic data (output of `RumoPreProcessor`), and, if available, the previously computed `RumoTrainsAhead` output file. The module separates dates for which engineered features have not yet been computed (i.e. new data), and modularly computes each of the feature categories described in Section 3.2.5. The code is organized to enable flexibility and customizability. The newly computed features are concatenated to any previously computed features passed to the module. This becomes one of the inputs to `ModelBuilder`.

4.4.2 Prediction and Evaluation

Our models are trained to predict times for individual Locals, but we perform evaluation on full train trips: series of adjacent Locals with a total length of 6 to 12 or 12 to 24 hours. The `TripGenerator` module takes a list of Locals, along with how much time was spent in each, and returns all contiguous subsets of Locals which make up a trip of the desired size.

The `ModelBuilder` module uses all of the above modules along with user input to create training and test sets, set the target variable, and fit any user-supplied base model at the chosen leve lf granularity. It will save predictions made with multiple models on all of the target variables, and give Local-level and trip-level metrics for the predictions.

5 Conclusion

When traveling by train, you can rest assured that your trip arrival time typically won't deviate from the ETA by more than half an hour or so, but the story is very different with multi-day freight trains, which can be late by many hours. We were pleased to be able to improve upon Rumo's current simulation, but there is still much work to be done to meet the bold accuracy goals that will make a big impact for Rumo's operations.

We had to learn a great deal about rail logistics before being able to make informed design decisions for our approach, which was a challenging and fulfilling experience. The results we achieved are only the surface of our final product. In particular, we are excited to pass on to Rumo a robust and configurable model architecture—which enables a range of model designs we have only begun to explore—as well as set of reproducible engineered features useful for future models. Not only did we learn about rail networks and complex regression tasks, we also learned from experience a number of strategies and stumbling blocks for organizing and executing large projects, detailed in Section 6.

We hope that Rumo, and perhaps future NYU Center for Data Science Capstone groups, are able to continue pushing this complex task forward, and we hope to remain in touch to find out what comes next.

6 Learnings

We learned many technical and practical lessons while working on our project, and some of our most valuable lessons related to process and project management.

Rapid Prototyping: Our deep focus on data exploration, engineering features, and building out our highly flexible model architecture left less time than desired for running our model on large datasets and getting meaningful results. Early model results were often quite bad and featured heavy overfitting. Because our model was so flexible, it became difficult to tell how to iterate: we could change the underlying regression models or their hyperparameters, we could change the model's level of granularity, or whether it is predicting raw values or deviations from historical means—there were simply too many options to navigate in a principled way and too little time to experiment. We would have benefited from building a deliberately simple model very early and adding on complexity *as-needed*.

Better Learning from Bad Models: When poor model predictions arose, we took the opportunity to iterate. However, our early error analysis was inadequate to best determine what to change for future iterations. The aforementioned model complexity meant that there were many possible ways that things could go wrong, including bugs in our growing codebase. Tracking down the exact reason a model is underperforming can be difficult, but we could have more closely examined, for example: performance differences relative to trip time; feature importances and how removing the most important features would impact model performance; relative impact of deviations from normal in moving time and anomalies.

Finding Existing Patterns in the Data: Much of our data exploration was focused on understanding the format, quirks, bugs, and distributions of the features (and target variables) in the dataset. We didn't sufficiently explore the relationships between and among features and prediction targets. Instead, we primarily relied on our models to toss out the unimportant features. Thus we missed out on building a richer understanding of which features were valuable or predictive, which were misleading, and which could potentially be made more useful via a transformation. If we were to begin the data exploration process again, we would likely seek out features that are correlated with each other and with the targets. Understanding linear and nonlinear relationships in the dataset would inform our data transformation and modeling process.

Build off of Existing Work: We learned a lot from Rumo about their current simulation-based method for forecasting train times. However, our mentor had also built several machine learning models for the task that we did not get details about until late into the process. Our mentor's best-performing model was a convolutional neural network predicting at the *trip* level, which is a radically different approach from the one we took. We may have gone down a different path if we'd had that discussion earlier. Furthermore, we discarded much of the research into transit time prediction methods we found, largely due to the particulars of freight train prediction not aligning with the tasks in published papers like bus or subway time prediction. However, there was likely more we could have taken inspiration from in the literature on transit time estimation even though these problems did not align perfectly with Rumo's task.

Domain Understanding Pays Off: We benefited greatly from our conversations with Rumo regarding how the train network functions. Understanding particulars like how single track networks lead to trains waiting in yards while others cross, and the circumstances under which one train will get priority over another, was crucial when it came to engineering features to encode relevant network information. Once we understood how the relationship between the length of a train and the length of a yard could determine whether or not a train would receive priority to cross, we asked for data on the length of each section of track and Rumo provided fairly comprehensive data on just that. It would have been beneficial to obtain similar domain understanding in the area of anomalies. Time spent waiting due to an anomaly comprises 20.9% of total trip time, on average, but exactly what an anomaly *is* and the factors that lead to how long they will take remained relatively unclear to us. Anomalies are, by definition, difficult to predict, but we likely could have learned more from Rumo and possibly obtained more relevant data on how and when anomalies arise.

Codebase Organization: While our modeling approach was complex nearly from the start, we still made many changes to it as we progressed. One side-effect of this was that our code evolved in a somewhat tangled fashion, often without larger deliberate architectural decisions dividing up logical pieces. Better planning and continuous refactoring could have improved the code quality, readability, maintainability, and likely decreased bugs, but we were not able to prioritize it. Investing earlier in a more deliberate architecture design would have reduced the need for rewriting and reorganizing later.

Acknowledgments

We would like to thank Rafael Orsoni, our mentor at Rumo, for his guidance throughout the project. Rafael gave us just the right balance of autonomy and instruction, and we can't thank him enough for his responsiveness, openness, and patience. Thanks also to Camila Durigan at Cosan for the thoughtful dialogue and big-picture perspective, Rafael Pinto for early support, and the entire team at Rumo and Cosan for the opportunity to work on this project.

References

[1] William Barbour, Juan Carlos Martinez Mori, Shankara Kuppa, and Daniel B. Work. Prediction of arrival times of freight traffic on us railroads using support vector regression. *Transportation Research Part C: Emerging Technologies*, 93:211–227, 2018.

[2] Charul and Pravesh Biyani. To each route its own ETA: A generative modeling framework for ETA prediction. *arXiv e-prints*, art. arXiv:1906.09925, June 2019.

[3] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. doi: 10.1145/2939672.2939785. URL `http://doi.acm.org/10.1145/2939672.2939785`.

[4] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 3146–3154. Curran Associates, Inc., 2017.

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] Thilo Reich, Marcin Budka, Derek Robbins, and David Hulbert. Survey of ETA prediction methods in public transport networks. *CoRR*, abs/1904.05037, 2019. URL `http://arxiv.org/abs/1904.05037`.

Appendix

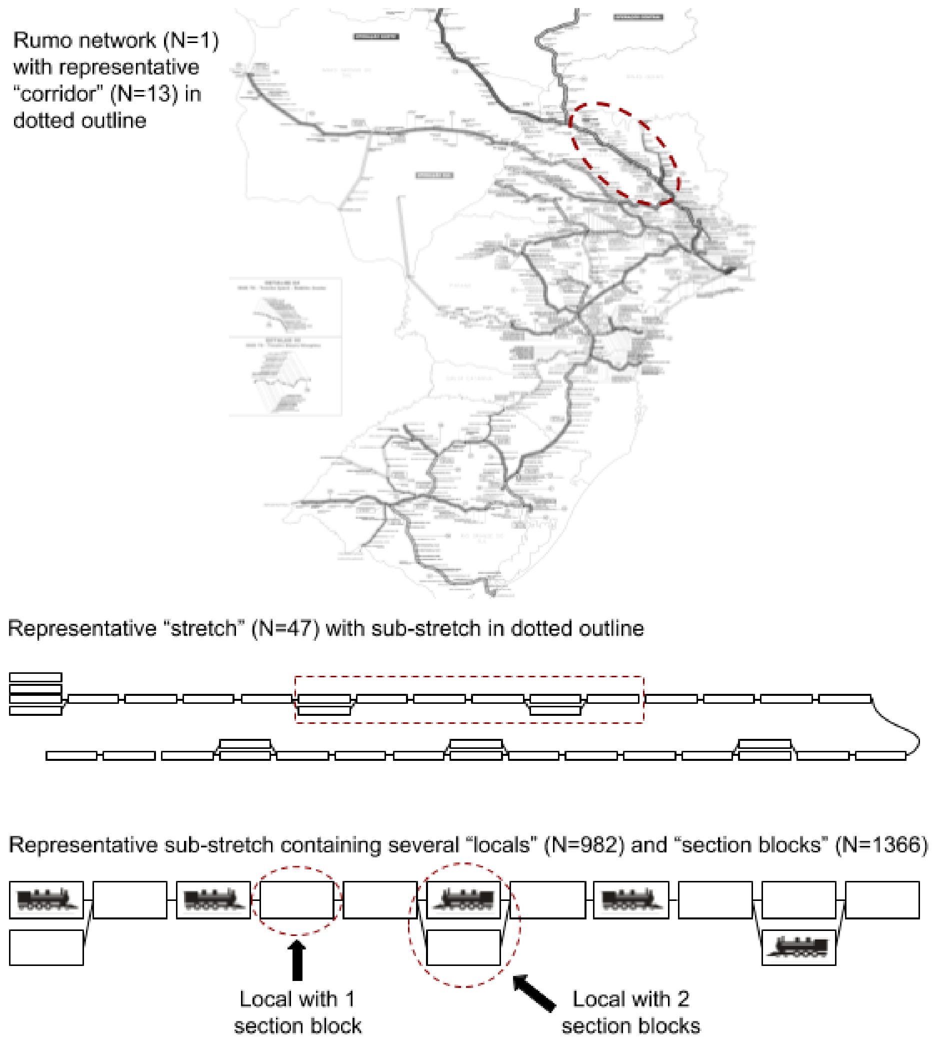

Rumo network (N=1) with representative "corridor" (N=13) in dotted outline
Representative "stretch" (N=47) with sub-stretch in dotted outline
Representative sub-stretch containing several "locals" (N=982) and "section blocks" (N=1366)
Local with 1 section block
Local with 2 section blocks

Figure 4: Rumo railway network hierarchy.

Field	Definition
Trains Ahead	
ta_same_dir1_03	# of same-direction trains in the next 3 Locals at time ts_exit
ta_same_dir1_05	# of same-direction trains in the next 5 Locals at time ts_exit
ta_same_dir1_10	# of same-direction trains in the next 10 Locals at time ts_exit
ta_same_dir1_99	# of same-direction trains between current Local and destination at time ts_exit
ta_opp_dir1_03	# of opposite-direction trains in the next 3 Locals at time ts_exit
ta_opp_dir1_05	# of opposite-direction trains in the next 5 Locals at time ts_exit
ta_opp_dir1_10	# of opposite-direction trains in the next 10 Locals at time ts_exit
ta_opp_dir1_99	# of opposite-direction trains between current Local and destination at time ts_exit
ta_same_dir2_01	# of same-direction trains in the next 1 ($\geq$ 2 capacity Locals) at time ts_exit
ta_same_dir2_02	# of same-direction trains in the next 2 ($\geq$ 2 capacity Locals) at time ts_exit
ta_same_dir2_05	# of same-direction trains in the next 5 ($\geq$ 2 capacity Locals) at time ts_exit
ta_same_dir2_99	# of same-direction trains in ($\geq$ 2 capacity Locals) between current and destination
ta_opp_dir2_01	# of opposite-direction trains in the next 1 ($\geq$ 2 capacity Locals) at time ts_exit
ta_opp_dir2_02	# of opposite-direction trains in the next 2 ($\geq$ 2 capacity Locals) at time ts_exit
ta_opp_dir2_05	# of opposite-direction trains in the next 5 ($\geq$ 2 capacity Locals) at time ts_exit
ta_opp_dir2_99	# of opposite-direction trains in ($\geq$ 2 capacity Locals) between current and destination
Crossings Ahead	
xa_future_n_14	mean # of future crossings for trips with this prefix in the last 14 days
xa_future_n_60	mean # of future crossings for trips with this prefix in the last 60 days
xa_future_t_14	mean total future crossing time for trips with this prefix in the last 14 days
xa_future_t_60	mean total future crossing time for trips with this prefix in the last 60 days
xa_rate_n_14	(# of crossings encountered so far) / (mean total for this prefix over the last 14 days)
xa_rate_n_60	(# of crossings encountered so far) / (mean total for this prefix over the last 60 days)
xa_rate_t_14	(running total crossing time) / (mean total for this prefix over the last 14 days)
xa_rate_t_60	(running total crossing time) / (mean total for this prefix over the last 60 days)
ya_future_n_14	mean # of future yards for trips in the last 14 days
Yards Ahead	
ya_future_n_60	mean # of future yards for trips in the last 60 days
ya_future_t_14	mean total future yards time for trips in the last 14 days
ya_future_t_60	mean total future yards time for trips in the last 60 days
ya_rate_n_14	(# of yards encountered so far) / (mean total for this prefix over the last 14 days)
ya_rate_n_60	(# of yards encountered so far) / (mean total for this prefix over the last 60 days)
ya_rate_t_14	(running total yard time) / (mean total for this prefix over the last 14 days)
ya_rate_t_60	(running total yard time) / (mean total for this prefix over the last 60 days)
Stretch Congestion	
tc_current	# of trains in the current trecho
tc_next	# of trains currently in the next visited trecho
tc_current_14	tc_current / (mean tc_current for this trecho over the last 14 days)
tc_current_60	tc_current / (mean tc_current for this trecho over the last 60 days)
tc_next_14	tc_next / (mean tc_next for this trecho over the last 14 days)
tc_next_60	tc_next / (mean tc_next for this trecho over the last 60 days)

Table 2: Selected engineered feature definitions.