Applying Explainable AI to Taxi Driver Classification

Major Qualifying Project

Authors:
William Burke
Ian Coolidge
Jin Ryoul Kim

Advisor:
Yanhua Li, Ph. D

24 March 2023

Contents

1 Introduction **3**

2 Background **4**
2.1 Dataset 4
2.2 Recurrent Models 5
2.2.1 Varying Sequence Length 6
2.2.2 Long-Short Term Memory Networks 7

3 Methodology **8**
3.1 Data Processing Pipeline 8
3.1.1 Describing the Dataset 9
3.1.2 Data Selection 9
3.1.3 Data Cleaning 10
3.1.4 Feature Engineering 12
3.2 Training the Deep Learning Model 14
3.2.1 Model Architecture 14
3.2.2 Training Process 15
3.3 Explainable Methods 16
3.3.1 Tabular Feature Importance 16

4 Results and Conclusion **17**
4.1 Model Results 17
4.2 Explanation Results 23
4.2.1 Takeaways 24
4.2.2 Future Work 25
4.3 Conclusion 25

1 Introduction

The world has been undergoing rapid urbanization and the placement of better technology in urban environments has allowed for a step towards more advanced cities. Nowadays technology is becoming increasingly necessary in the lives of individual residents, with smartphones putting readily-available information about traffic, safety, health services, and community news into millions of hands. Furthermore our ability to measure, store, and analyze data on large scales has increased the capacity to which information can be leveraged in countless domains, including smart cities.

Machine learning has become a crucial tool for smart cities. It is able to consume huge amounts of data generated by running and monitoring urban centers with the help of technologies such as the internet of things becoming more common. Machine learning models can process this data, producing actionable results to be used to reach solutions to several urban problems. For example, it can be used to propose solutions related to resource management, traffic patterns, infrastructure, or really any problem backed by sufficient data.

However, as data becomes more complex and more powerful deep learning architectures are used, the methods used to reach a solution become much less interpretable on the surface level. For example, when utilizing the power of a deeper model like a neural network, it's common to either reach a solution or a shortcoming without fully understanding which components and patterns of the data lead to that outcome.

In recognition of such an issue, more developers are starting to incorporate explainable AI into their complex solutions in order to make these problems more understandable. This places more emphasis on human understanding, which is certainly crucial to a problem due to the degree of complexity that the most powerful solutions introduce. Furthermore, our research aims showcase the application of explainable AI to a complex and hard to interpret solution in the urban intelligence domain, specifically related to traffic. In this paper, we build a deep classification model with the goal of predicting the identity of a driver responsible for a taxi trip, train it on complex data, and use explainable AI to better interpret the results of this model.

2 Background

2.1 Dataset

We used a dataset of taxi trips representing the rides given by 442 different taxi drivers over the course of 1 year. Additionally, this data was collected in the city of Porto, Portugal between July 2013 to June 2014. Our group chose this dataset due to the versatile features it includes, as well as the fact that there are more than enough entries to reliably train a model. More specifically, each data sample corresponds to one completed trip, containing a total of nine features. Most notably among these features is a list of gps coordinates representing the taxi's trajectory for that trip, which we refer to as the trip's 'polyline'.

Polyline. The polyline field drew the largest amount of attention from us, as it is the only field that can be described as spatial-temporal data, and is by far the most complex. The polyline is a sequential list of gps coordinates, with these gps coordinates being measured at 15-second intervals in the format (longitude, latitude). The longitudinal component is used interchangeably with the "x" component, which fell into the range of [-5, -13] for this dataset, and the latitudinal component is used interchangeably with the "y" component, which fell into the range of [38.5, 42.5]. Because it is a sequential list, the first coordinate pair represents the trajectory's start location and the last coordinate pair represents its end location. The visual below is an example of a trip's polyline from our dataset, visualized over the actual location using OpenStreetMaps and Matplotlib.

Other Fields. The 8 remaining fields in the dataset are the following, as described by the dataset's publisher:

- TRIP_ID: A string containing a unique identifier for each trip.
- CALL_TYPE: A char from 'A', 'B', 'C' used to identify the method of demanding the trip's service.
  - A: Trip was dispatched from the central
  - B: Trip was demanded directly to a taxi driver on a specific stand
  - C: Otherwise (i.i. a trip demanded on a random street)
- ORIGIN_CALL: An integer uniquely identifying the phone number which was used to demand, at least, one service. Identified only if the CALL_TYPE is 'A'
- ORIGIN_STAND: An integer uniquely identifying the taxi stand. Identifies the starting point only if the CALL_TYPE is 'B'
- TAXI_ID: An integer containing a unique identifier for the taxi driver that performed each trip. This was the target class of our classification problem.

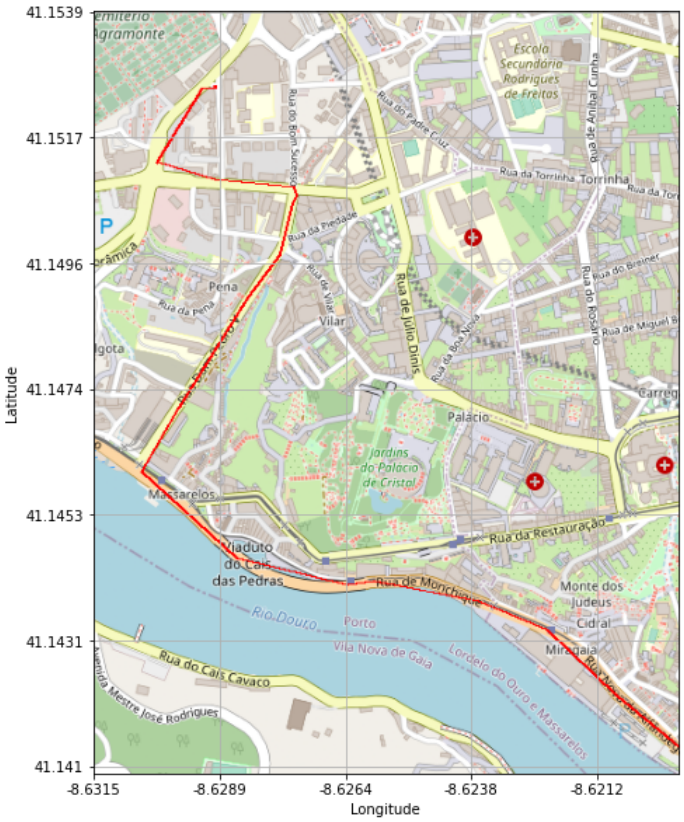

Figure 1: Visualizing a polyline in the city of Porto, Portugal.

- TIMESTAMP: An integer representing the unix timestamp in seconds that identifies the trip's start time and date.
- DAY_TYPE: A char that identifies if the trip was started on a holiday, day before a holiday, or otherwise. This field was largely unpopulated, so we were not able to make much use of it.
- MISSING_DATA: A boolean, false when the GPS data stream is complete, true if one or more locations are missing.

2.2 Recurrent Models

Recurrent neural networks, or RNNs, are neural networks designed to process sequential data in which information computed from previous values of a time series can impact the model's prediction on future values. RNNs differ from traditional, or feedforward, neural networks in the addition of a *recurrence*: some quantity that is propagated forward through the network from cell to cell.

Recurrent neural networks are a natural choice for processing sequential data due to their ability to assemble predictions based on data propagated from previous features. In the case of the polyline taxi trajectory data set, recurrent networks can retain information about previous points in the trajectory while processing future points.

Figure 2 shows the basic structure of an RNN, illustrating the recurrence concept. Note that there are various implementations for the prototypical recurrent cell R_j, and each cell produces an output H_j. Depending on the implementation of the overall network, some recurrent models may only make use of the final output H_n, while others make use of every output H_j. In the case of our model, we generally choose to only use the final output H_n, where n is the overall length of the polyline.

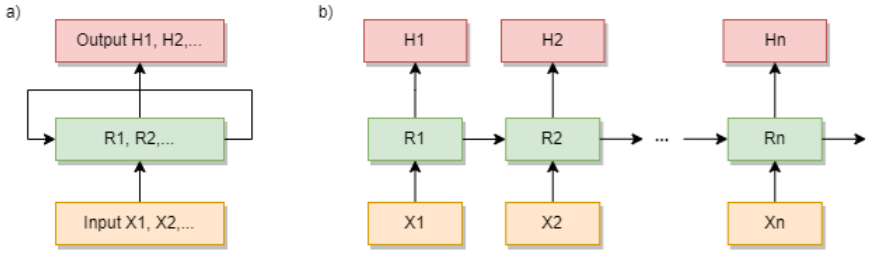

Figure 2: Prototypical structure of a recurrent neural network. Part a) shows the repeated recurrence, and part b) demonstrated the "unrolled" RNN.

2.2.1 Varying Sequence Length

One particularly useful feature built into many RNN architectures is the fact that they use a single cell repeated an arbitrary number of times. This means that these RNNs can take input sequences of varying lengths, rather than constraining their input to batches of fixed-size samples. In the case of the polyline dataset, the length of a polyline is directly proportional to the total ride time, and various rides will have different lengths.

While RNNs are generally capable of handling these various lengths during training, they are unable to handle batches of sequences of different lengths. To remedy the problem and train the RNN on varying sequence lengths, several options are available:

- Find the maximum length N of all sequences, and pad all sequences in the training set with zeroes or the mean to length N. This has the disadvantage of potentially losing information about each sequence's length.
- Create batches in such a way that all sequences in each batch have the same length, e.g. by sorting sequences by their length. This is the most flexible option, but can lead to inconsistent batch sizes. In addition, for

the polyline trajectory problem, it is likely that few polylines will have the exact same length leading to a small maximum batch size.

- Constrain the batch size parameter to 1. This is the simplest option, but potentially increases training time by requiring many batches per training epoch.

For simplicity and understandability, in our experiments we chose to constrain the LSTM batch size to 1. In addition to causing a moderate slowdown in training speed, this constraint can lead to suboptimal local minima; however, due to the complexity of the other options and the difficulty of dividing by polyline length, we found this option to be both relatively simple and relatively powerful compared to the other two solutions.

2.2.2 Long-Short Term Memory Networks

LSTMs, or Long-Short Term Memory networks, are one of the most common and powerful implementations of the RNN architecture. LSTMs attempt to simulate human memory via the implementation of a "forget gate". The forget gate is used to mitigate the effect of the vanishing and exploding gradient problems, in which the long-term sequential nature of an RNN leads to gradients that become excessively small or excessively large. Via the ability to learn how to forget information that came from sufficiently earlier in the sequence, LSTM networks provide a solution to the vanishing and exploding gradient problems.

3 Methodology

The methodology of our research question can be categorized into three sections, which include the data processing pipeline, training the deep neural network, and the use of explainable AI techniques to make the model more interpretable. In the diagram below, their abstract relationship is outlined, which more specific details shown in the respective sections. The first aspect is our data processing pipeline which consists of steps executed to transform the original dataset into one that fits our needs of feeding samples with meaningful features into the model. Next, we trained a deep neural network with the major internal mechanism being an LSTM layer for the polyline features, which feeds into the final layer that also makes use of the remaining tabular features. Finally, we explored a series of explainable AI techniques with the aim of making this deep model more interpretable, which typically involved retraining and extracting information from the training process.

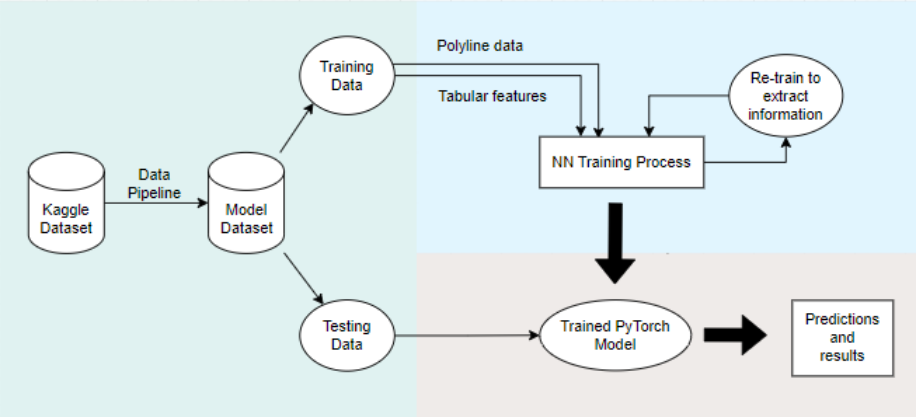

Figure 3: Diagram showing components of methodology.

3.1 Data Processing Pipeline

As mentioned above, this section with outline the steps taken to transform the raw Kaggle dataset to the final dataset that is used in our neural network's training and testing processes. This pipeline starts with data selection to narrow the number of drivers to 5 to fit our classification problem. Next, we clean the dataset to remove unusable data or extreme outliers, including geographical filtering. Finally, we do feature engineering to create new features based on polylines and tabular data related to the trip, where we also transform certain features so they can seamlessly be fed into our machine learning model.

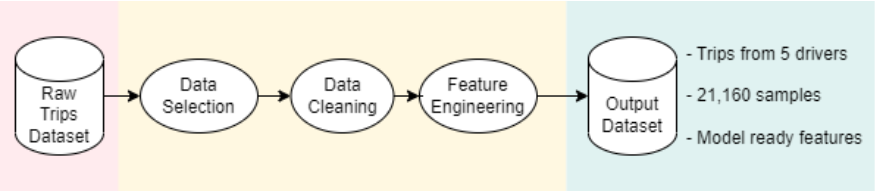

Figure 4: Diagram showing data processing pipeline.

3.1.1 Describing the Dataset

After downloading our dataset from Kaggle, we had to do an initial analysis to better understand the contents of the data before actually shaping and selecting it to fit our needs. One of the actionable insights we gained from doing this was a distribution showing the number of rides given by the drivers represented in the dataset. We discovered that the majority of drivers had given between 3,000 and 6,000 rides, and that a small minority gave below 1,000 or over 8,000 rides. This information would prove very useful to our data selection process.

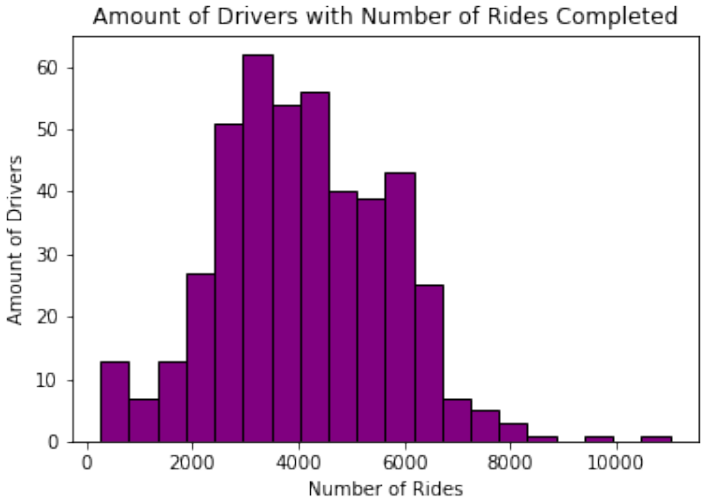

Figure 5: Histogram showing totals of drivers across number of rides given

3.1.2 Data Selection

The goal of our model was to take a sample taxi trip and use it to predict which driver gave that trip. Since it seemed impractical to make a prediction model based on hundreds of possible drivers, and therefore hundreds of classes, we decided that our research question could be answered with a subset of 5

drivers. When selecting these 5 drivers, we wanted to choose 5 that had similar representation in the data, meaning that they had given a similar, and reasonable, amount of rides. Therefore, we chose 5 drivers that had around 4,000 rides given, landing them in the middle of the distribution shown in the previous section. This was done to ensure that a selected driver would not vary too much from the standard we set on what is a "normal" driver, while also eliminating the possibility of a class imbalance when training the classification model. To recap, we selected a subset of trips distributed roughly equally across 5 drivers, while ensuring that each driver had a similar, reasonable amount of rides given.

3.1.3 Data Cleaning

After doing data selection to narrow our working sample to 5 drivers and their respective trips, the next step was to make sure the data set was clean. We did this by removing any 'outlying trips' which we defined as any trip without a polyline, or any trip with a poly outside of a defined geographical square. Although the vast majority of the data was kept, this process ensured that some samples that could have skewed the results of our analyses or model were removed.

The first process of dropping records that had outlying polyline fields was relatively straightforward. Since the polyline field is one of the main areas of interest in our research question, it was necessary to remove any trip that did not have a polyline, despite what data the other fields contained or why the trip did not have a polyline. Expanding upon this, we decided to remove any polyline that had less than 5 points, because that polyline length would indicate that the trip lasted 1 minute or less, and was likely an outlier. This process was relatively straightforward, as we could simply drop any records where the length of the polyline field was less than 5, which ended up being a very insignificant portion of the dataset at less than 0.01%.

The data-cleaning process of removing polylines outside a defined geographical square was more complex, as it required more technical steps and careful consideration of the points removed. The motivation behind a geographical square was to create an upper and lower boundary for latitude (geographic Y coordinate) and longitude (geographic X coordinate), which contained the majority of polylines in the samples of 5 drivers, while also filtering out extreme outliers. For example, an outlier trip may be extended into the ocean due to a measurement error, extend out of the country, or simply go far outside of the geographic area we decided to focus on.

The first step in this process was to visualize the distribution of points, both for their latitude and longitude. This would allow us to see what an outlier looked like on the lower and upper end of both dimensions, so we could then set a reasonable upper and lower boundary for each dimension to filter out outliers. The following visuals show the distribution of longitude and latitude for points contained in polylines in the dataset.

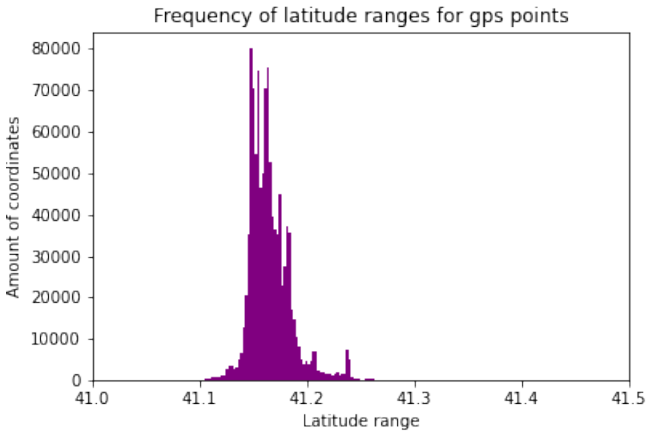

Figure 6: Figure showing distribution of latitude of points

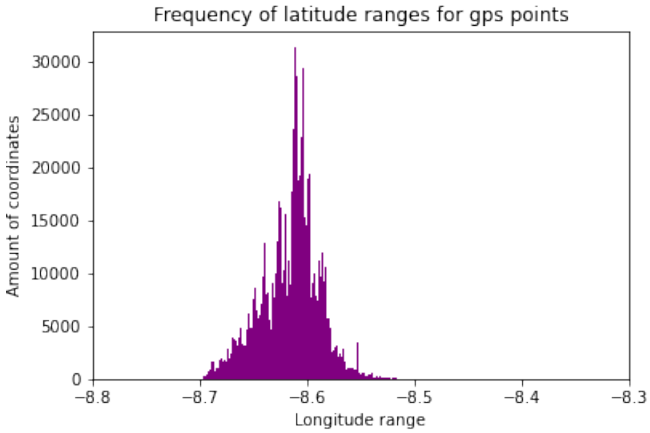

Figure 7: Figure showing distribution of latitude of points

We used these visuals to set the upper and lower boundaries for latitude and longitude, ultimately using a latitude range of [41.1, 41.3] and longitude range of [-8.7, -8.5]. From there we flagged any trip with a polyline containing at least one point outside these limits, allowing us to filter the dataset to remove any flagged trips. Below using OpenStreetMap, we were able to visualize the geographical region contained within these boundaries.

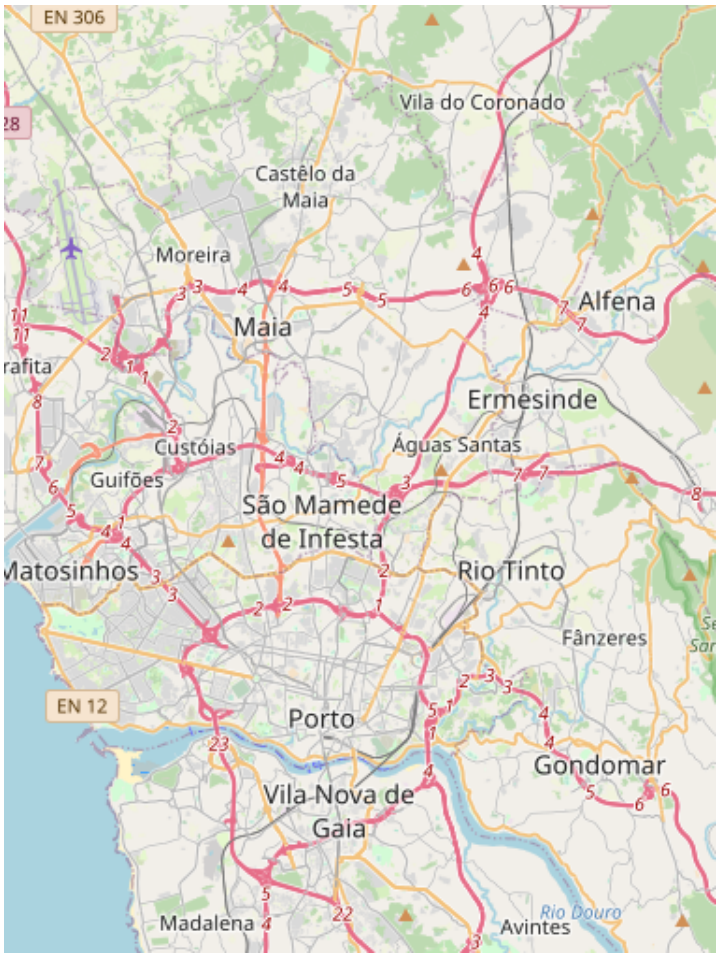

Figure 8: Region containing all points after data cleaning.

This entire process of data cleaning allowed us to remove 579 outlying samples that could have impacted our results. This left us with 97.4% of the uncleaned dataset of 5 drivers, or 21,660 samples to be analyzed, transformed, and used for modeling.

3.1.4 Feature Engineering

The final step of our data processes was feature engineering, where we created more meaningful features based on the baseline set of features we had, and transformed some of the features so they could be better interpreted by our neural network. These features include a normalized polyline, start, end, and various other coordinates, polyline length, distance traveled, average distance between points, call type one hot encoded, day of week one hot encoded, and

transformed time of day.

Normalized Polyline. One of the first processing steps we took was to normalize all coordinate points contained in polylines on a [-1,1] scale. The motivation for this was to scale down the longitude and latitude numbers so they could be interpreted by the model more easily, with the added benefit of making the data more interpretable and workable to us. Since the normalization of points requires a minimum and maximum in each direction, we were able to use the boundaries of the geographical square that we used in the data-cleaning step. Furthermore, after this step, it was more appropriate to refer to points as 'x' and 'y', rather than longitude and latitude. The resulting polylines containing normalized 'x' and 'y' coordinates were then used for the feature engineering of any polyline-related features, as well as being the input for the LSTM component of our model.

Single Point Features The first feature we extracted from the normalized polylines was the start location of the trip. The motivation behind this was that the geographical location of where a driver started their trip could potentially be an insightful piece of information that could help the model differentiate the drivers. Following this logic, we later decided to incorporate the first 5 points of the polyline, in addition to the last coordinate of the polyline representing where they ended their trip. Additionally, when feeding a single point into the model (ex: start location), we had to split it into separate 'x' and 'y' features, meaning that we had 12 features total representing these points relative to the driver's path.

Whole Polyline Features The next features we engineered from the polyline were its length, representing how long the ride took, the distance covered, and the average difference between points. Since each polyline's data points were collected exactly 15 seconds apart, the length of a polyline is directly correlated with how much time the trip took. For this feature, we simply took the number of coordinate pairs in each polyline, which ranged from 7 to around 200. Additionally, we wanted to predict a close estimate of how much distance was covered by a trip because intuitively, the distance covered in a trip could factor into a prediction of who was driving. To do this, we calculated the distance between each connected pair of points in the normalized polyline and summed them together, resulting in a number that closely, but not perfectly, represents the physical distance traveled by the car. Finally, an average delta feature was calculated by computing the average distance between points, as opposed to sum of distances in the previously mentioned feature.

Categorical features. The next set of features that we needed to prepare for the model were the categorical features that were provided by the dataset. The only categorical feature in the original kaggle dataset was call type, with a value of 'a', 'b', or 'c', which we one hot encoded resulting in one feature for each of the possible categories.

Datetime features. Another one of the features provided by the Kaggle dataset was a DateTime object representing the date and time of the start of the trip. First, we wanted to incorporate the day of the week that the trip occurred into the model, which we did by extracting the weekday from the DateTime object, then one hot encoding it into 7 features representing the days of the week. Next, we extracted the time of day from the DateTime object, which required more transformation due to the cyclical nature of time of day. We transformed the time of day to a decimal representation, then applied sine and cosine and scaled it down to be between -1 and 1. This resulted in two features (with a minimum value of -1 and a maximum of 1) representing time of day, one for sine and one for cosine. This practice is common when trying to preserve the cyclical nature of certain fields like time because it preserves the fact that 23:00 is closer to 1:00 than 1:00 is to 4:00 etc.

3.2 Training the Deep Learning Model

3.2.1 Model Architecture

Our model consists of two primary components: an LSTM operating on the polyline data, and a feedforward neural network operating on the tabular features extracted from the data as described in Section 3.1.3. The architecture of our neural network model is demonstrated in 9.

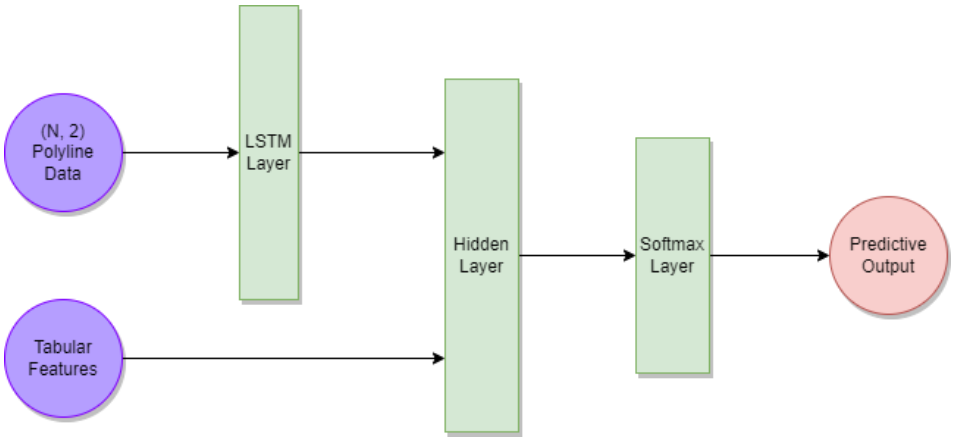

Figure 9: High-level overall neural network architecture

The recurrent section of the architecture is a single-layer LSTM with a hidden layer size of 200. Our experiments found that multi-layer LSTMs generally provided small to non-existent performance increases over a single-layer LSTM at the cost of significant performance, and as a result we chose to constrict our experiments to a single LSTM layer. We choose to only consider the output of the final unit of this LSTM layer.

The remaining neural network architecture consists of a 20-dimensional vector corresponding to the various tabular features effecting the network's performance. This vector is concatenated with the two-dimensional output of the LSTM and fed into a 100-dimensional feedforward network layer, followed by a 5-dimensional softmax layer corresponding to the five drivers. By definition of the softmax function, the output of our model is a 5-dimensional probability distribution consisting of likelihoods that the polyline belongs to a given driver.

This model notably contains many simplifications which improve training performance at the cost of complexity in the model which may improve overall accuracy. These include, but are not limited to:

1. Using only a single LSTM layer rather than a deep multi-layer LSTM.
2. Only considering the final output of the LSTM layer, rather than every intermediate output or a subset of the intermediate outputs.
3. Only including one ReLU-activated hidden layer before the softmax output layer.
4. Using relatively small (100-200) feedforward layer sizes.

We performed several experiments on our model in order to see the effect of relaxing any of these conditions, however none of these experiments led to results that performed noticeably better than the simple model. Given the limited computing power of the WPI Turing Cluster, we thus chose to use one of the simplest possible models to produce our predictions.

3.2.2 Training Process

We minimized a cross-entropy loss over our model's output. For each training period as described in Section 3.3.1, we trained for 100 epochs. We used an Adam optimizer for adaptive training, using default PyTorch hyper-parameters and a learning rate of 5e-5. This learning rate was chosen via a train-validation split with all features, with 10 percent of the training set used for validation. Via this validation, we found that changes to other hyperparameters had minimal impact on the model's overall performance.

Our training and feature importance calculation was performed on a single GPU on the **WPI Turing Cluster**, with a job allocated the following specs:

- 8 GB RAM
- NVIDIA A100 GPU
- 8-hour time allocation

We found 8 hours of training sufficient to train each model involved in the tabular feature importance process.

3.3 Explainable Methods

3.3.1 Tabular Feature Importance

Our primary goal in explaining the results of the model described was to determine the relative contributions of the various model inputs, including the polyline LSTM, tabular features provided by the dataset, and features extracted manually from the polylines. We define a feature of the model to mean some subset of the data columns. For example, the "Start Location" feature consists of the x and y coordinates of the start location, and thus consists of two columns. Given N features we extracted an importance score for each feature via the following process.

First, we train the model as normal with all features according to the training process described in Section 3.2.2, obtaining a percent accuracy A_{control}. For each feature $\phi = \{c_1, \ldots, c_\ell\}$ that we wish to compute an accuracy score for, we then retrain the model after shuffling each of the columns c_j that is part of the feature. This yields an adjusted accuracy score A_ϕ for each feature ϕ, with the expectation that $0 < A_\phi < A_{\text{control}}$ for each feature ϕ.

To compute the feature importance score I_ϕ for each feature ϕ, we then apply the following formula:

$$I_\phi = \frac{A_{\text{control}} - A_\phi}{A_{\text{control}} - \frac{1}{c}}$$

Here c represents the number of classes, and thus $\frac{1}{c}$ is the expected accuracy of a fully random classifier. This score represents the percentage loss in accuracy resulting from shuffling the feature ϕ. The calculation has the property that, assuming A_ϕ ranges from $\frac{1}{c}$ to A_{control}, the feature importance score will lie in the interval $[0, 1]$. In particular, if $A_\phi \approx A_{\text{control}}$, then removal of the feature ϕ does not significantly impact the accuracy and $I_\phi \approx 0$ as expected. Similarly, if $A_\phi \approx \frac{1}{c}$, then removal of the feature ϕ reduces the accuracy to near zero in which case $I_\phi \approx 1$ as expected.

This process and analysis shows that, by computing I_ϕ for each feature over the course of $N + 1$ training applications of the model, we obtain relative importance indices for each feature.

4 Results and Conclusion

4.1 Model Results

Overall, our trained model correctly classified about 58.4% of samples in the testing set. However, being a multi-class classification problem, there is more information beyond overall accuracy to show more specifically how the model performed and where it fell short. As a result, we used the previously trained PyTorch model to make predictions on the test set, and saved those results containing the actual and predicted class of every sample trip. With this, we were able to do an analysis and got the following results.

The first, and most basic piece of information beyond overall accuracy is accuracy per class, which is the percentage of samples correctly classified for each class. As shown by the table below, some classes have a relatively high proportion of correctly classified samples (around 70%), and others are unusually low (49.5% and 26%). This is an interesting result, as it suggests that the model was really accurate when classifying drivers 1, 3, and 4, but did a very poor job at predicting drivers 0 and 2.

Table 1: Percent correctly classified.

Driver 0	**Driver 1**	**Driver 2**	**Driver 3**	**Driver 4**
49.5%	76.7%	26.0%	69.2%	73.4%

Another one of the results we derived from the dataset of predictions was the amount of times that class was predicted, compared to the number of trips that actually belong to that class. As shown in the visual below, class 2 was the least frequently predicted class, while simultaneously having the highest actual amount of samples in the testing set. This is very strange because although it had the most training examples to train the model, it still was guessed at a significantly lower amount than any of the other classes. Another point of information worth noting is that the actual class distribution is relatively even, while the model's predicted class distribution is less so with class 2 having significantly fewer predicted values. This would suggest that the model was trained to slightly over-predict classes 0,1, 3, and 4, while severely under-predicting class 2.

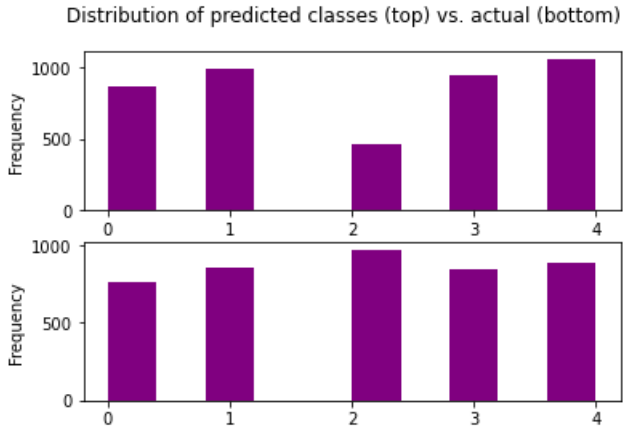

Figure 10: Predicted classes (top) and actual classes (bottom) on test set.

Next, we used a confusion matrix to gain a more specific breakdown into the class-by-class predictions. This confusion matrix below shows the specific count of predicted vs. actual guesses for each pair of classes. This matrix shows more detail about how class 2 was under-predicted, and where that class's samples were distributed. Notably, 0 was predicted as the class value when the actual class was 2 400 times, exceeding the number of correct classifications for class 2 by 147 samples. This is very interesting because as the only major inconsistency in the confusion matrix, it suggests that classes 0 and 2 had a similar feature space that would cause this large of a discrepancy.

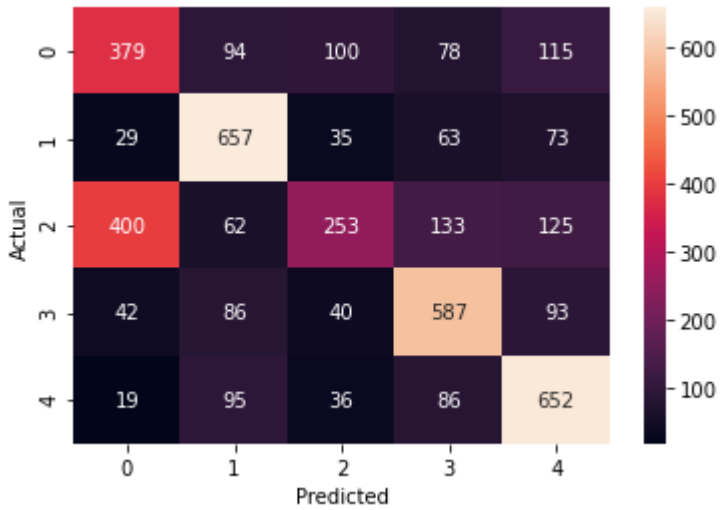

Figure 11: Confusion matrix showing predicted and actual class confusion.

So far, the information has revealed that not all classes were predicted with the same success as others, which is valuable information in itself. It shows that the model was significantly more accurate for certain classes compared to others, which raises the question of what patterns in the input data or lack thereof caused this. This is inherently due to the patterns data that were fed into the model to represent each of these classes, which we can begin to take a look at. To elaborate on this, we will compare key geographical features of two classes that were not confused at all, and two classes that were confused often.

First, we'll compare classes 0 and 2, which were two of the most confused classes. As seen in the confusion matrix, they were confused a total of 500 times (400 and 100). Looking at the features that were fed into the model, one can begin to understand why the classes were not easily differentiable. In the following series of graphs, plotted on a normalized x-y geographical plain, all polylines or start locations belonging to a driver are displayed, separated by correctly classified and incorrectly classified samples. To the human eye, it's clear that there is a minor difference between the correctly classified samples as expected, while the incorrectly classified samples appear to have very similar information.

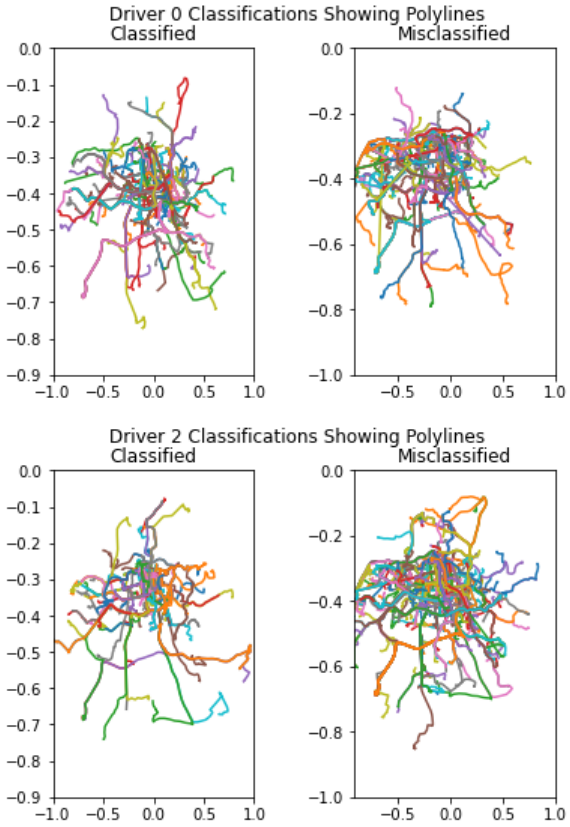

Figure 12: Graphing classified and misclassified polylines on normalized x-y plane.

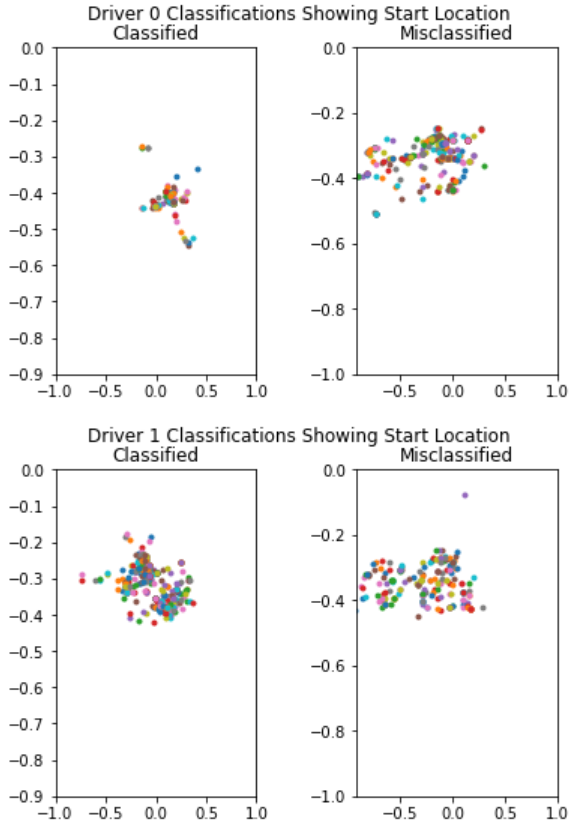

Figure 13: Graphing classified and misclassified start locations on normalized x-y plane.

Next, we'll compare two of classes that were not mistaken nearly as much as others, classes 0 and 1. As you can see in the confusion matrix above, classes 0 and 1 were confused for each other a total of 97 times (62 and 35), which is very small compared to other classes. Contrary to the previous example, one can begin to see a significantly greater difference in properly classified samples while the misclassified samples still reveal little information, but show a more unique shape compared to the previous example.

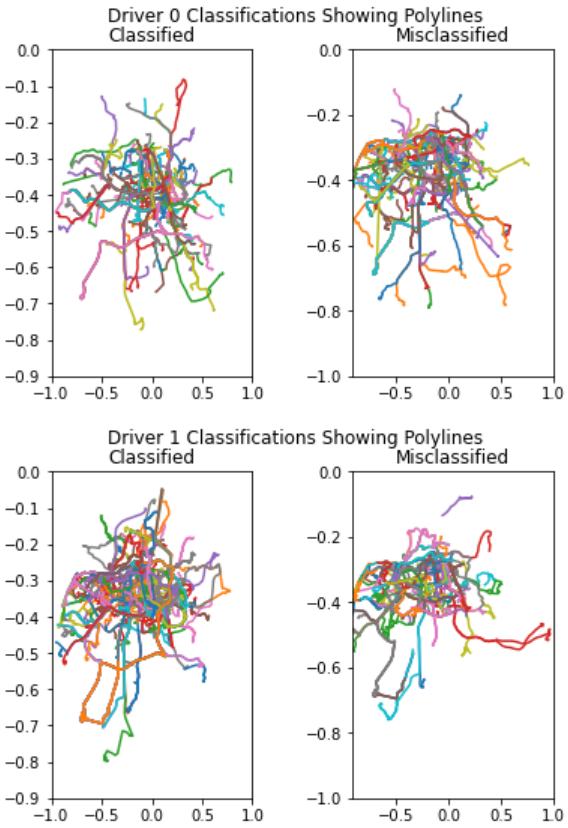

Figure 14: Graphing classified and misclassified polylines on normalized x-y plane.

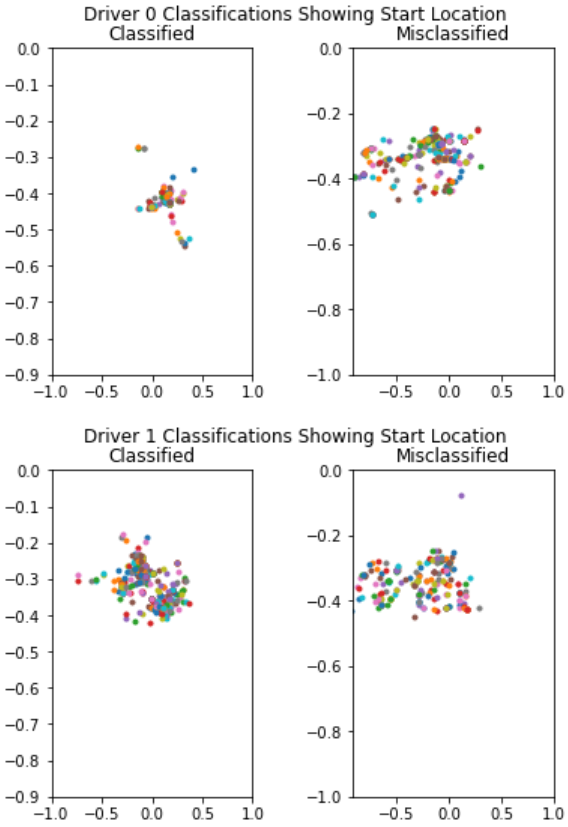

Figure 15: Graphing classified and misclassified start locations on normalized x-y plane.

4.2 Explanation Results

The overall testing accuracy of our control network was $A_{control} = \mathbf{0.5836}$. Using this baseline value we were able to compute a feature importance index for each feature. These are tabulated in 2. The calculation of I_ϕ for each feature was done according to the computation in Section 3.3.1.

Our tabular feature importance calculation returned the following accuracy values and feature importance scores for each feature:

Feature	Accuracy A_ϕ	Feature Importance I_ϕ
call-type	0.4776	0.2763
day-of-week	0.5238	0.1559
time-of-day	0.5392	0.1157
distance-traveled	0.5258	0.1507
speed	0.5302	0.1392
start-loc	0.4548	0.3358
end-loc	0.5602	0.0610
mid-loc	0.4335	0.3913
polyline	0.4610	0.3196

Table 2: Accuracy and feature importance values for each feature ϕ.

4.2.1 Takeaways

We noted minimal to statistically insignificant improvement in performance for certain features leading to low feature importance scores. This demonstrates that our model was able to identify features that were useful predictors of the taxi driver, as well as features which had little relevance to the overall output. In particular, the end location, speed, and time of day features had negligible importance.

Due to the high feature importance scores for start location and middle location, we expect that the end location feature is highly correlated with these features, meaning it provides little additional contribution to the overall model. We were surprised to find that the time of day had little impact on the model's output, and the likely conclusion from this is that the five drivers we arbitrarily chose tended to give rides at similar times of day. Finally, the lack of importance for speed is relatively expected for two reasons: first, speed is directly correlated to other features such as distance traveled and the length of the polyline, and second, in an urban environment it is relatively unlikely that there will be high variations in traffic speed. This is particularly true in the case of strict traffic laws that taxi drivers may be required to follow in this city.

We were encouraged to see that the polyline feature, which was the LSTM output, had a relatively nontrivial feature importance comparable to several of the other highly correlated features. Future research would be needed to determine whether this is due to the polyline containing trivially obtainable information such as the start and end locations of the ride, or whether the model gleaned nontrivial information from the LSTM layer. Either way, this is an encouraging result that shows that sequential data can be successfully processed through recurrent models.

Further, we noted significant variance in the ability of the model to accurately predict drivers depending on the driver in question. This suggests that certain drivers have distinct route patterns, while others may be more difficult to identify. This makes intuitive sense. Due to the large amount of data discarded from our dataset, future research on this dataset should consider attempting to train the model on other sets of drivers to see if the performance is better, or

worse, than the dataset we extracted.

4.2.2 Future Work

This project represents a significant body of work towards understanding sequential data. In particular our primary conclusion that tabular features had a significantly larger contribution to the output than the polyline itself is a hypothesis that can be tested further. In order to improve the overall results of our model as well as provide a more convincing explanation of our features, we provide several improvements that could be made to our model and training process in the future.

1. Evaluating the performance of the LSTM model with a more complex recurrent neural network, such as a multi-layer or stacked LSTM.
2. Applying sequence-level feature engineering on the polyline, e.g. computing the differences between subsequent polyline points and using them as supplemental inputs to the LSTM.
3. Applying state-of-the-art sequential saliency mapping techniques to understanding the polyline LSTM as a whole. Due to the fact that the LSTM overall had a low feature importance score, any saliency mapping on the LSTM would have yielded statistically unimportant results, but this may help future research to understand the LSTM's poor performance.
4. Utilizing outputs from the LSTM other than the final output h_N.
5. Resampling the data to select a different combination of 5 drivers, or a different number of drivers.
6. Utilizing other methods to allow mini-batches of size greater than 1 during training, which would lead to more consistent convergence of the model and less random variance in the final output accuracies.
7. Including regularization and dropout in our model to prevent overfitting, particularly in the relatively complex LSTM layer.

4.3 Conclusion

In conclusion, our project was able to obtain strong correlations between polyline features and taxi driver classes. Our results show that a combination of recurrent processing on the polyline along with tabular feature extraction creates a model with better overall performance than either of these components individually. Further, by applying a linear tabular feature importance calculation, we were able to discern differences in importance between the various tabular features including the polyline itself. This has allowed us to develop a rich understanding in the features that induce differences between the taxi drivers in question.

Our work lends itself to future work, in which future research could involve delving into the polyline itself to determine which aspects of the recurrent model have significant impacts on the output.

References

[1] Brownlee. J. (2016). *Sequence Classification with LSTM Recurrent Neural Networks.* Retrieved from: https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

[2] Jose. G (2019). *Predicting Sequential Data using LSTM.* Retrieved from: https://towardsdatascience.com/time-series-forecasting-with-recurrent-neural-networks-74674e289816

[3] Rojat. T., Puget. R., Filliat. D. (2021). *Explainable Artificial Intelligence (XAI) on Time Series Data.* Cornell University; arXiv.

[4] Srivastava. P. (2020). *Deep Learning: Introduction to LSTM.* Retrieved from: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/