Machine Learning: Course Project

Olga Chernytska

June, 2018

Topic
Predicting bus time to arrival at stops using GPS data only.

Topic Importance and Application
Many European cities have electronic scoreboards on every bus stops that show minutes to arrival by transport routes. It makes public transport more convenient. But such scoreboards are not so widely used in Ukraine. As at now, the best prediction of time to arrival is provided by EasyWay on their website/application [1]. However, their prediction system is based on Google Maps statistics and sometimes on average speed on particular route interval. Google Maps may not count for number of stops on the route, stop time or traffic light schedule, while average speed approach is rather static and is too 'average'.

My aim is to build model using GPS data. The results of the model can be used for developing electronic scoreboards on the bus stops as well as by EasyWay to improve their time to arrival predictions.

Data
The data used is collected by friend of mine, Maksym Hontar, using tool [2]. Data contains information on 8 distinct routes for Lviv city that pass Ukrainian Catholic University. Time period: since 2017-07-18 to 2018-05-28. The main fields are:
– datetime request time
– routecode, routeid, routename
– state 1 if bus is on route
– vehicleid, vehiclename
– x, y longitude and latitude of the vehicle at the time of request

Request is made every minute; response returned contains all the vehicle coordinates that are currently on route. Time of request is assumed to correspond to time when vehicle is in the location returned in response. However, it is not always true - data can be sent with a lag. Lags are random and can be of any length. Unfortunately, I cannot deal with it any way.

For analysis I selected route A53 (Id: 1723724) as it has information on movements of 36 distinct vehicles, which is about 3.1 millions of data points spread during 11 months.

There are two more data files:
– coordinates of stops for route 1723724 split by direction;
– coordinates of route points for route 1723724 split by direction.

Methodology

I slightly modified method, explained in [3, p.356]. Authors propose to predict travel time from location c to stop j (T_{cj}) using the following formula:

$$T_{cj} = T_{ij} - T_{ic}$$

where T_{ij} is predicted travel time between stops i and j, and T_{ic} - time that bus already passed from stop i to its location c. T_{ic} is determined by subtracting departure time at stop i from the time when bus is on location c. T_{ij} is determined using artificial neural networks. Originally, the following features are used: X_1 - time of the day, X_2 - code for station i, X_3 - code for station j, X_4 - travel time from 0 to i. The output Y is predicted bus travel time to reach stop j from stop i - T_{ij}.

My modifications are the following:

– Model predicts travel time only between nearest stops. To predict time to arrival to stop that is far away, I sum up predictions for the travel time between all the nearest stops, that are on the path.

– Instead of artificial neural network, I used linear regression as baseline model and gradient boosting as improved one.

– I used the following features: season (winter, spring, summer, autumn), weekday, hour, minute (split by 15-minute intervals) and route interval (characterized by starting and ending stops). Time features were extracted from the time stamp when bus is on the starting stop. All features were converted to dummy variables.

Preprocessing

The data contains GPS coordinates only. Model inputs are season, weekday, hour, minute and route interval. Model output and Y's for training - travel time between nearest stops. So some preprocessing steps has to performed.

Step 0. When split by month, it can be seen that path for route 1723724 changes in October-December 2017, but than comes back to normal (see Figure 0). So these months are excluded from analysis.

Step 1. Find all continues sequences 'time - coordinates' when bus is on the route. First, sort data frame by vehicle id and datetime. Find sequences of 1's ($state = 1$ means that bus is on the route). Label first data point in each sequence as 'start', last data point as 'end' and all data points in between as 'on route'. Let's call this sequence as a drive. And let's give a unique id to each drive. Figure 1 shows how data looks like after this step.

Step 2. Select only full drives and assign them to directions ('there' and 'back', or 0 and 1). Full drive means, that drive starts and ends near the starting and ending stops for particular route and direction. To do this, I found the starting and ending stops coordinates for the routes and directions (coordinates for 4 distinct stops in total).

For every drive I checked whether data point labeled as 'start' is near the start stop for direction 0 or direction 1 ('near' means no further than 300 m away). I name the data point as 0/1, or -1 – if it is far from all start stops. The same - for the ending stops. As a result, for each drive I gained labeling 0/1/-1 for each 'start' and 'end' data points. Figure 2 shows how data looks like after this step.

Step 3. At the previous step, I gained labeling 0/1/-1 for each 'start' and 'end' data points. Then I split drives by the directions. If 'start' and 'end' data points for the particular drive correspond and equal to 0, we know that this drive is full drive in the direction 0; and for direction 1 - respectively. There were a lot of partial drives, but after this step, all the dirty data was excluded from analysis.

Clean data (only full drives) was shifted to new data frames: one data frame contains drives in the direction 0 and another - in direction 1. Example of the data frame for each direction is shown in the Figure 3.

Step 4. Remove drives, that have big intervals between two nearest data points. Some drives that are recognized as full data is so sparse that it is impossible to understand how bus moved. These drives we should exclude to preserve data quality.

Step 5. Add artificial data. We are adding starting and ending stop coordinates (as observations) for those drives that start/end not near starting/ending stops. We assume that bus was in these locations 15/30/45 seconds earlier/later depending on how far the current drive start/end from true stops.

Data is sparse, if no errors we receive bus locations once in a minute. Now it is hard to understand what exact time bus was near the stop. We want to decrease this interval to 15 seconds, by adding 3 middle points between each pair of points for the drive.

Step 6. Remove duplicated locations at the drive ends. Sometimes state of the bus is 1 (which means that it is on route), but actually it is staying on the starting/ending stop with no change in location. For starting point we want to leave just the last one duplicate, because that represents time when the bus starts its move to next stop. For ending stop - opposite; we leave just the fist duplicate as it is the time when bus ends its move from previous stop.

Step 7. Find closest coordinate on the drive to each stop. Using stops coordinates, for every direction and drive we can find at what exact date stamp bus was near the particular stop. We are looping through the every drive, looking for all data points that have minimal distances to all stops. But algorithmically, it is done a bit more efficiently. As a result, we have approximated time when the bus was near the every stop performing particular drive. Result are show in Figure 4.

Step 8. Rearrange data. In this step we rearranging data by calculating time in seconds between every two nearest stops for every drive. Results are shown in Figure 5.

Step 9. Test data. Firstly, we want to exclude drives for which algorithm couldn't find all the stops. This is important because time to move between stops are calculated incorrect in this case for all the intervals on the route. Secondly, route points that were recognized as closest to stops are highly spread around actual stop coordinates. We want to clean it as well by introducing a threshold - maximal distance by which coordinates for observations can be far away from actual stop coordinates. Small threshold excludes more data points, big threshold leaves more dirty data. We selected threshold such way that most of the stops have non-overlapping surroundings and no more than quarter of observations is lost.

Now data looks better but still there are issues with couple of stops. Issues may occur because of dirty data on stop coordinates for the route. For instance, buses that moves in the direction 0, do not start on the 'true' starting point (highest one on the figure). This issue makes it impossible to predict time between start point and the next one.

The second problem that number of observations decreased unevenly for different route intervals. Now we have not so may observations for several intervals that may decrease the quality of prediction.

Figure 8 and Figure 9 show distribution of time to move between nearest stops. Data still has outliers, but most of the observations are located in narrow range close to median.

Modeling

Two data sets - for directions 0 and 1 were concatenated by introducing new variable - 'direction'. This is done to train a single model, but not two models for every direction. Data set contains 150,864 data point.

Time variable contains important features, which were used to train a model - season (winter, spring, summer, autumn), weekday, hour and minute (by 15-minute intervals, four levels in total). Starting and ending stops were merged to be a single variable – 'route interval'. All the explanatory variables were converted to dummy variables; first level was excluded to prevent linear dependence of features. As a result, there are 57 explanatory variables. Variable to predict - travel time between every nearest stops, in seconds.

All the observations were randomly split into train and test set, 75% and 25% observations respectively. Metric to evaluate model was chosen to be median absolute error, because data set contains a lot of outliers, but we are interested more to fit non-outliers.

Baseline model is linear regression. Its median absolute error on test set is 30.86 seconds. The first improvement was done by excluding outliers from train set. Outliers are observations that lie beyond time travel intervals [mean - std; mean + std]. Intervals are calculated for every direction and route interval. As a result, 6.8% of train set observation identified as outliers and excluded from train set. Test set outliers were not excluded. This approach lead to decrease in median absolute error on train set to 29.28 seconds. Was decided to use train set without outliers as for linear regression it showed lower error rate.

Gradient Boosting Regressor (`sklearn` package) showed better result with tuned parameters. Parameters were tuned according to procedure recommended by DataRobot [4]:

– Pick 'n estimators' as large as computationally possible.

– Tune 'max depth', 'learning rate', 'min samples leaf', and 'max features' via grid search.

– Increase 'n estimators' even more and tune 'learning rate' again holding the other parameters fixed.

The actual results of Grid Search (`GridSearchCV` from `sklearn`) are the following:

– Number of estimators was fixed to 300. Grid Search was performed through parameters 'learning rate': [0.1, 0.05], 'max depth': [4, 6, 8], 'min samples leaf': [5, 9, 11]. The best combination - 'learning rate': 0.1, 'max depth': 6, 'min samples leaf': 11 - results in median absolute error of 19.93 seconds on cv set. It should be mentioned that cross validation was performed on train set with outliers excluded.

– As best 'min sample leaf' was selected to be 11, which is upper bound, I decided to additionally perform grid search through parameters: 'min samples leaf': [11, 14, 17, 20]. 'Learning rate' and 'max depth' were fixed at the level selected previously (0.1 and 6, respectively). Best 'min samples leaf' is still 11.

– Number of estimators was increased to 1000; 'max depth' = 6, 'min samples leaf' = 11. Search was performed through 'learning rate': [0.1, 0.5, 0.02, 0.01]. The best learning rate is 0.1. Median absolute error on cv set is 20.08 seconds.

Final model is Gradient Boosting Regressor with parameters: 'n estimators' = 1000, 'learning rate' = 0.1, 'max depth' = 6 and 'min samples leaf' = 11. Median absolute error on test set is 21.77 seconds, which is 40% decrease compared to baseline model.

Errors in various 'direction - route interval' vary a lot. Figure 10 and Figure 11 shows some groups with 25% higher and 25% lower group median error than total median error (21.77). The major issues are with 'direction = 0, route interval =#26 - #27', where median absolute error is 108.14 seconds, and with group 'direction = 1, route interval #27 - #28', which median error is 88.73 seconds.

Demonstration

The task is to predict time to arrival to next 5 stops using time, direction and bus GPS coordinates [latitude, longitude] as inputs.

For demonstration purposes, I implemented function that randomly generates inputs - time, direction and GPS coordinates. Direction is a random integer - 0 or 1. Time is randomly selected from the data set available. GPS coordinates are generated from the route points data. I preproccesed route points data, so it is possible to get a point on every location on the route.

On the first step, algorithm identifies between what nearest stops bus is located. Location is represented by previous and next stops. Additionally, algorithm outputs share of interval between these stops that bus already passed (variable 'passed'). At most 5 next stops are returned as well.

On the second step, algorithm predicts time to arrival (in seconds) using model developed in previous section. Generated inputs are converted to appropriate format that model accepts - all the variable are converted to dummies with the same names as in train set. Travel time is calculated for every nearest pair of stops, time to arrival is calculated as cumulative sum for all the previous stops.

Predictions are done in the following way:

– To calculate travel time to first nearest stop, inputs are: previous and next stops, time and direction. The output is multiplied by (1- 'passed'), to subtract time that bus already passed from previous stop. Here travel time is equal to time to arrival to first nearest stop, which we predict.

– To calculate travel time from first nearest stop to second nearest stop (and for further route intervals), inputs are: first nearest and second nearest stops, time (but adding travel time calculated on previous step) and direction. I decided to update time because it can take 20 and more minutes when bus reaches last predicted 5th stop, but during this time rush our may end or start, so travel time will be different. To calculate time to arrival to particular stops, algorithm cumulatively adds all the travel times to previous stops.

Algorithm outputs data frame that contains 5 next stops that bus is going to pass, seconds to arrival and actual arrival time for every stop.

Demonstration can be found on the Figure 13 and in Jupyter Notebook 'ML Project - Demonstration'.

Conclusion and Further work

Throughout this project the following tasks were completed:

– Data was transformed from data frame 'time - vehicle id - latitude - longitude' to 'drive id - starting stop - ending stop - travel time in seconds'.

– Transformed data was used to train a model. Linear regression was selected to be a baseline model; it results in median absolute error of 30.86 seconds on test set. Baseline model was improved by 40% based on median absolute error decrease to 21.77 seconds on test set. This was done by excluding outliers from train set, changing model to be Gradient boosting regressor and tuning its parameters using Grid search.

– Developed working demo, that accepts bus GPS location, direction and time as inputs, and produces data frame that contains seconds to arrival and arrival time for the next 5 stops on the route.

The model was developed using only route id = 1223724. However, most of the code is standardized and can be easily converted to work for other routes. Demo works fast, because it loads and uses previously trained model, so it can be used in real time for future predictions. Median error of 21.77 seconds may be insignificant for customers - bus passengers.

However, accuracy of the model can be increased. Significant improvements lie in how data collected and processed. Now data is collected every minute, decreasing this interval to 15 seconds makes it possible not to use artificially generated midpoints. Approximation algorithm, that is expected to identify time when bus passes particular stop, works with errors which can be evaluated from Figure 6 and Figure 7. The other issue is mistakes in stops locations. Empirically, it was explored that buses moving in the direction 0, do not pass stop 0.

Another idea is to add variable that represent how the bus moved through previous route intervals during this drive. For instance, low speed on the previous route intervals may indicate that working day is a holiday.

As at now, model uses direction as an input. However, original data does not contain this variable. There are ways to deal with this problem: 1) use another source of data, for instance, EasyWay API; 2) develop algorithm that will identify direction; sequence of locations should be used as inputs in this case, because it is impossible to identify direction based on just single location.

References:

1. EasyWay. https://www.eway.in.ua/ua/cities/kyiv
2. City Transport/Routes

   Links: 82.207.107.126:13541/SimpleRIDE/Lad/Sr/,

   82.207.107.126:13541/SimpleRIDE/LODA/Sr/
3. Wei Fan, Zegeye Gurmu. Dynamic Travel Time Prediction Models for Buses Using Only GPS Data

   Link: https://www.sciencedirect.com/science/article/pii/S204604301630168X
4. Mark Steadman. Gradient Boosted Regression Trees: DataRobot blog.

   Link: https://blog.datarobot.com/gradient-boosted-regression-trees

APPENDIX A. Figures: Preprocessing Stage.

Figure 0. Raw data for route 1723724, split my month.

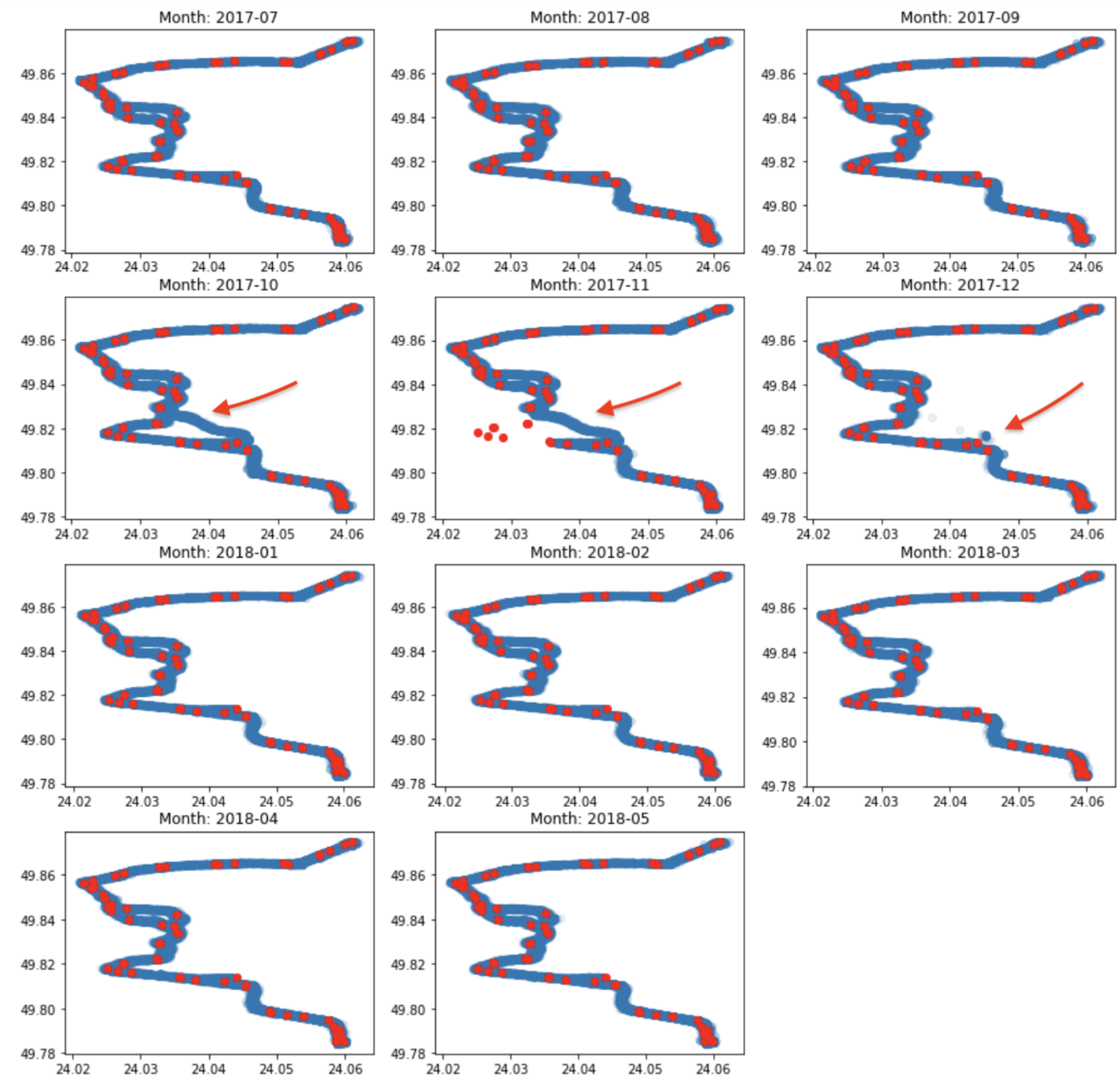


Figure 1. Results after Step 1.

datetime	vehicle_id	state	lon	lat		
2017-07-20 05:13:21.811899	37984.0	0.0	24.058700	49.790617	unknown	
2017-07-20 05:14:31.32854	37984.0	0.0	24.059367	49.784700	unknown	
2017-07-20 05:15:33.007255	37984.0	0.0	24.059883	49.785050	unknown	DRIVE
2017-07-20 05:16:22.452177	37984.0	1.0	24.059817	49.785450	start	
2017-07-20 05:17:21.934757	37984.0	1.0	24.059383	49.786133	on route	
2017-07-20 05:18:21.703389	37984.0	1.0	24.059183	49.787417	on route	
2017-07-20 05:19:31.559794	37984.0	1.0	24.059200	49.790567	on route	
2017-07-20 05:20:22.693668	37984.0	1.0	24.059067	49.791100	on route	
2017-07-20 05:21:31.948823	37984.0	1.0	24.058750	49.793283	on route	
2017-07-20 05:22:21.945059	37984.0	1.0	24.056267	49.795183	on route	
2017-07-20 05:23:22.38149	37984.0	1.0	24.053983	49.796300	on route	
2017-07-20 05:24:21.4714	37984.0	1.0	24.052783	49.796950	on route	
2017-07-20 05:25:22.271987	37984.0	1.0	24.048983	49.798917	on route	
2017-07-20 05:26:22.032683	37984.0	1.0	24.048883	49.798950	on route	
2017-07-20 05:27:21.796592	37984.0	1.0	24.046617	49.805717	on route	
2017-07-20 05:28:21.507506	37984.0	1.0	24.046833	49.808700	end	
2017-07-20 05:29:21.906135	37984.0	0.0	24.041800	49.811867	unknown	
2017-07-20 05:30:23.722561	37984.0	0.0	24.042267	49.812400	unknown	
2017-07-20 05:31:59.046659	37984.0	0.0	24.039600	49.813583	unknown	
2017-07-20 05:33:33.223789	37984.0	1.0	24.035483	49.813983	start	

Figure 2. Results after Step 2.

datetime	vehicle_id	state	lon	lat		
2017-07-20 05:13:21.811899	37984.0	0.0	24.058700	49.790617	unknown	
2017-07-20 05:14:31.32854	37984.0	0.0	24.059367	49.784700	unknown	
2017-07-20 05:15:33.007255	37984.0	0.0	24.059883	49.785050	unknown	DRIVE
2017-07-20 05:16:22.452177	37984.0	1.0	24.059817	49.785450	start	1
2017-07-20 05:17:21.934757	37984.0	1.0	24.059383	49.786133	on route	
2017-07-20 05:18:21.703389	37984.0	1.0	24.059183	49.787417	on route	
2017-07-20 05:19:31.559794	37984.0	1.0	24.059200	49.790567	on route	
2017-07-20 05:20:22.693668	37984.0	1.0	24.059067	49.791100	on route	
2017-07-20 05:21:31.948823	37984.0	1.0	24.058750	49.793283	on route	
2017-07-20 05:22:21.945059	37984.0	1.0	24.056267	49.795183	on route	
2017-07-20 05:23:22.38149	37984.0	1.0	24.053983	49.796300	on route	
2017-07-20 05:24:21.4714	37984.0	1.0	24.052783	49.796950	on route	
2017-07-20 05:25:22.271987	37984.0	1.0	24.048983	49.798917	on route	
2017-07-20 05:26:22.032683	37984.0	1.0	24.048883	49.798950	on route	
2017-07-20 05:27:21.796592	37984.0	1.0	24.046617	49.805717	on route	
2017-07-20 05:28:21.507506	37984.0	1.0	24.046833	49.808700	end	1
2017-07-20 05:29:21.906135	37984.0	0.0	24.041800	49.811867	unknown	
2017-07-20 05:30:23.722561	37984.0	0.0	24.042267	49.812400	unknown	
2017-07-20 05:31:59.046659	37984.0	0.0	24.039600	49.813583	unknown	
2017-07-20 05:33:33.223789	37984.0	1.0	24.035483	49.813983	start	

Figure 3. Results after Step 3.

drive_index	datetime	lat	lon
8	2017-07-20 06:29:23.173075	49.872750	24.059667
8	2017-07-20 06:30:24.701013	49.869133	24.056483
8	2017-07-20 06:31:53.319531	49.865000	24.052017
8	2017-07-20 06:33:40.596908	49.865100	24.051133
8	2017-07-20 06:34:33.000431	49.865350	24.046867
8	2017-07-20 06:35:33.301501	49.865050	24.042083
8	2017-07-20 06:36:37.595833	49.864483	24.037167
8	2017-07-20 06:37:29.130137	49.864150	24.035083
8	2017-07-20 06:38:28.338238	49.863383	24.032600
8	2017-07-20 06:39:32.822154	49.860300	24.027500
8	2017-07-20 06:41:22.883643	49.856917	24.021283
8	2017-07-20 06:42:27.378367	49.854450	24.022700
8	2017-07-20 06:43:22.513019	49.854250	24.023033

Figure 4. Results after Step 7.

	drive_index	datetime	lat	lon	closest_stop
0	8	2017-07-20 06:28:38.173075000	49.870820	24.057820	stop #00: id=36865
1	8	2017-07-20 06:29:23.173075000	49.872750	24.059667	none
2	8	2017-07-20 06:29:38.555059500	49.871846	24.058871	none
3	8	2017-07-20 06:29:53.937044000	49.870942	24.058075	stop #01: id=36853
4	8	2017-07-20 06:30:09.319028500	49.870037	24.057279	none

Figure 5. Results after Step 8.

	drive_index	start_stop	start_lat	start_lon	end_stop	end_lat	end_lon	start_time	end_time	time_diff_seconds
0	8	stop #00: id=36865	49.870820	24.057820	stop #01: id=36853	49.870942	24.058075	2017-07-20 06:28:38.173075000	2017-07-20 06:29:53.937044000	75.763969
1	8	stop #01: id=36853	49.870942	24.058075	stop #02: id=37283	49.869133	24.056483	2017-07-20 06:29:53.937044000	2017-07-20 06:30:24.701013000	30.763969
2	8	stop #02: id=37283	49.869133	24.056483	stop #03: id=36822	49.865100	24.051133	2017-07-20 06:30:24.701013000	2017-07-20 06:33:40.596908000	195.895895
3	8	stop #03: id=36822	49.865100	24.051133	stop #04: id=36823	49.865125	24.043279	2017-07-20 06:33:40.596908000	2017-07-20 06:35:18.226233500	97.629326
4	8	stop #04: id=36823	49.865125	24.043279	stop #05: id=36821	49.864908	24.040854	2017-07-20 06:35:18.226233500	2017-07-20 06:35:49.375084000	31.148851

Figure 6. Step 9: Dirty and clean data for direction=0.

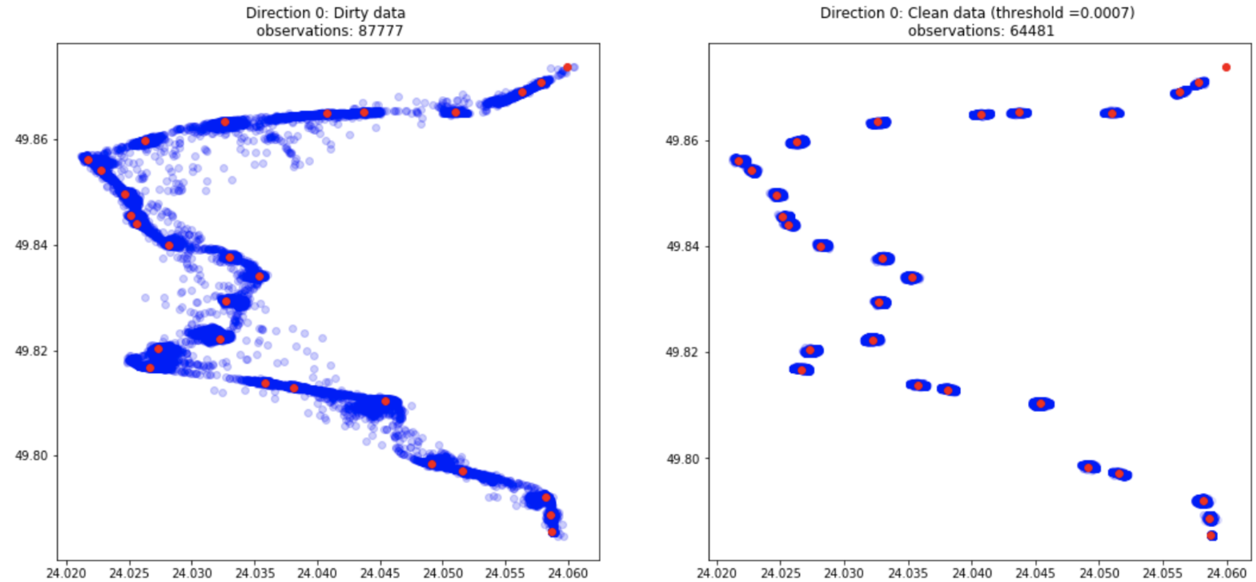


Figure 7. Step 9: Dirty and clean data for direction=1.

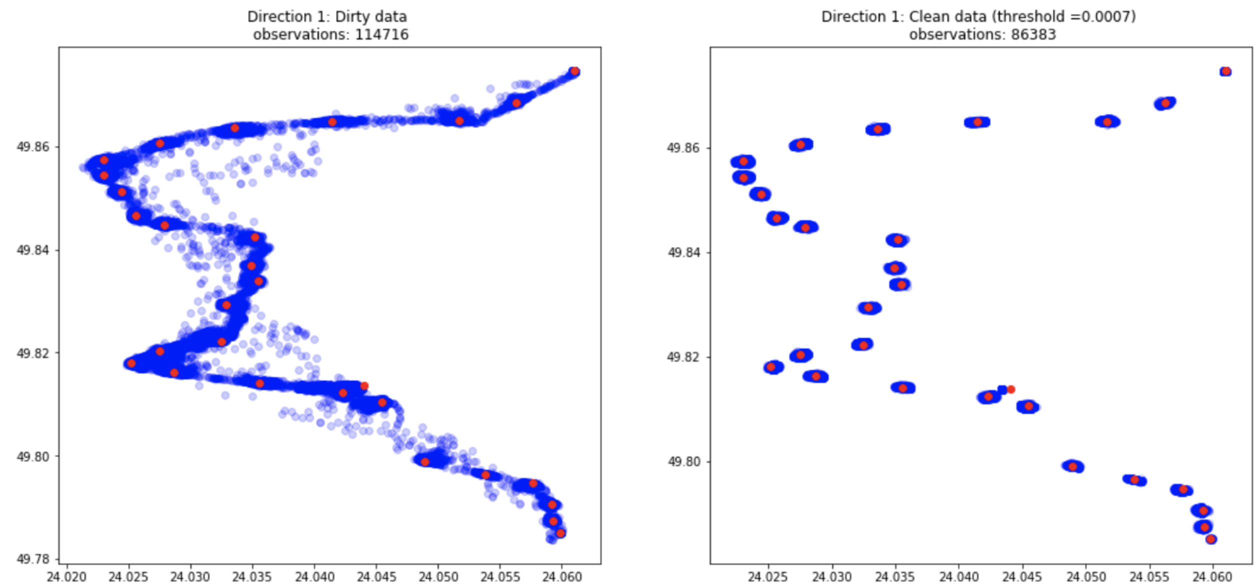


Figure 8. Distribution of time intervals: no zoom.

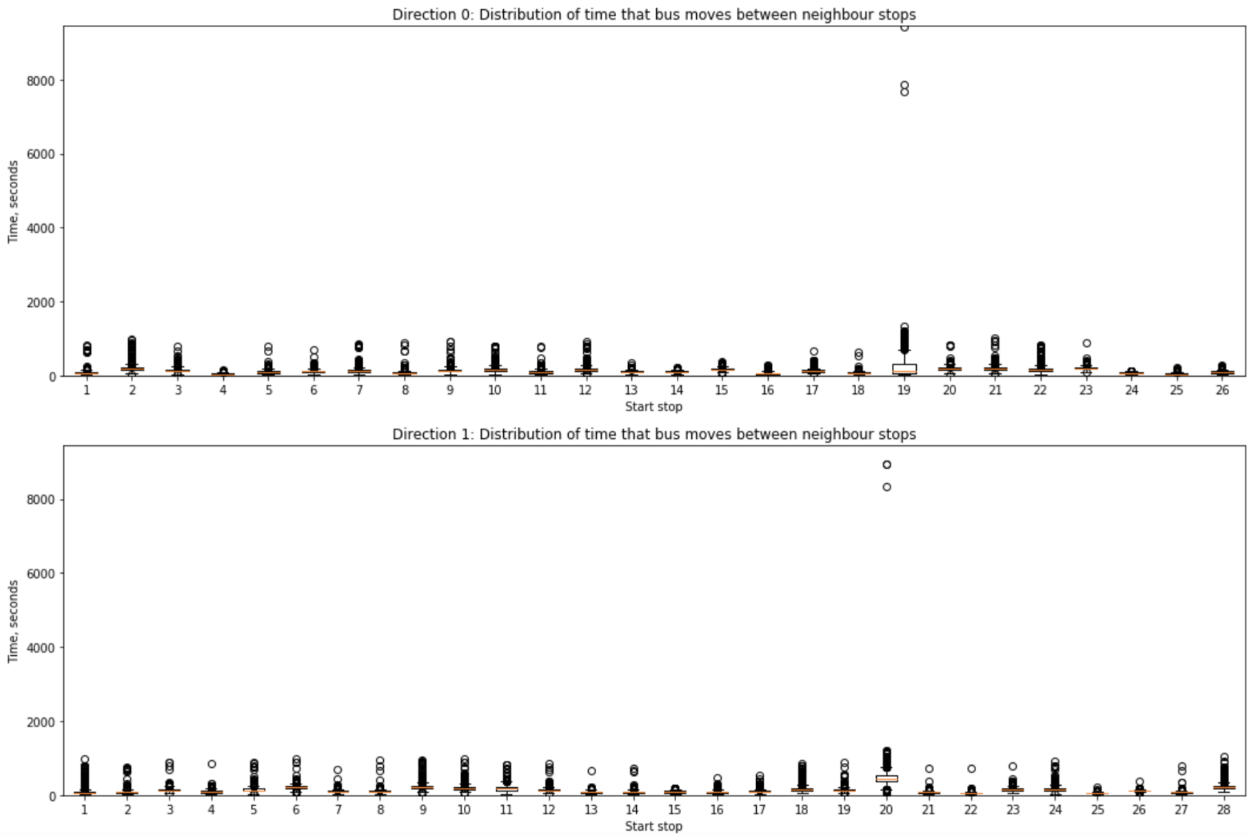
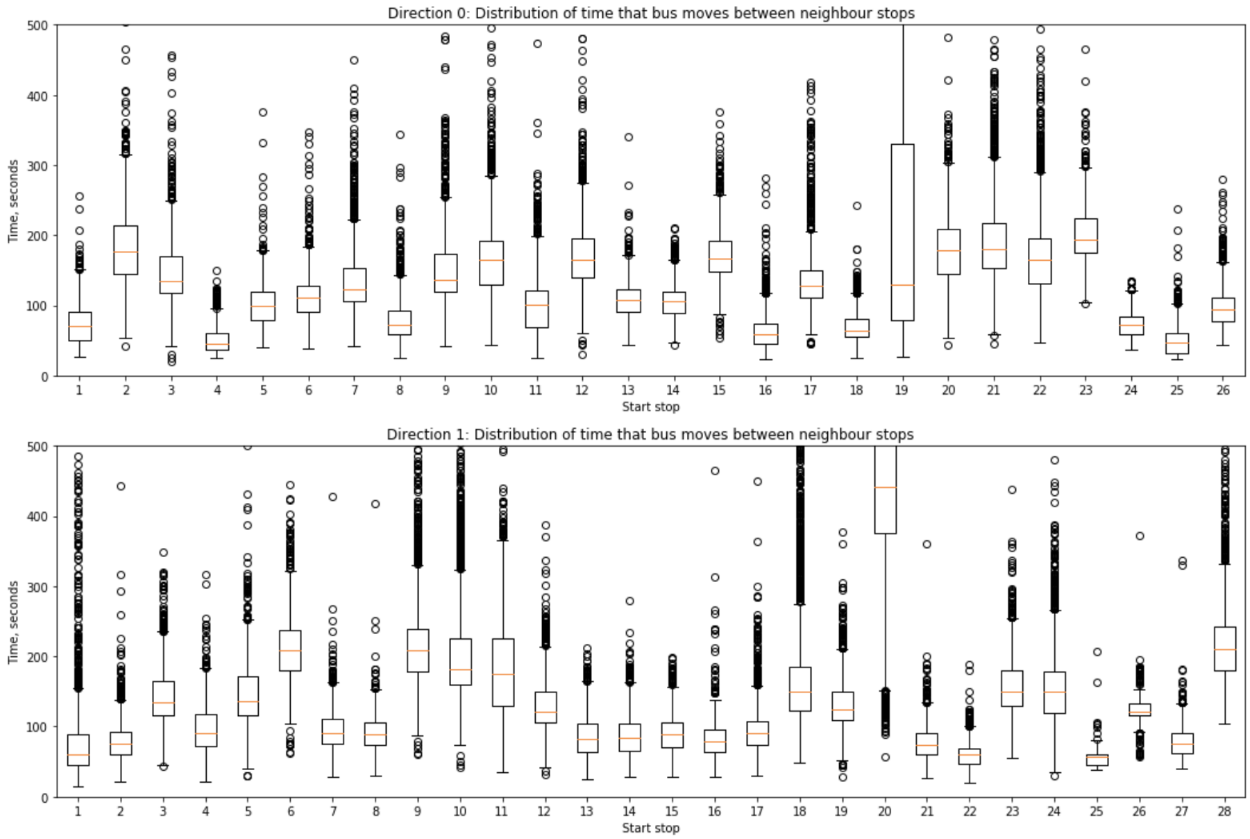


Figure 9. Distribution of time intervals: zoomed.



APPENDIX B. Figures: Modeling Stage.

Figure 10. Directions and route intervals with high median prediction error.

direction	route_interval	n	median_error	error_type
0	#26_#27	807	108.142714	high
1	#27_#28	924	88.727332	high
1	#18_#19	932	43.556659	high
0	#02_#03	657	33.289124	high
0	#12_#13	745	31.618724	high
0	#10_#11	748	30.619026	high
0	#15_#16	614	30.306155	high
1	#13_#14	452	29.956692	high
1	#17_#18	940	29.701877	high
1	#00_#01	1035	28.627297	high
1	#04_#05	935	28.508142	high
1	#16_#17	912	28.413712	high
1	#25_#26	834	28.261078	high
1	#05_#06	858	28.031559	high
0	#11_#12	806	27.417206	high

Figure 11. Directions and route intervals with low median prediction error.

direction	route_interval	n	median_error	error_type
0	#21_#22	418	16.309283	low
1	#01_#02	974	16.048940	low
1	#10_#11	634	15.712757	low
1	#23_#24	688	15.638464	low
1	#12_#13	417	15.081569	low
0	#19_#20	543	14.772386	low
0	#20_#21	571	13.429593	low
0	#25_#26	749	13.368453	low
0	#23_#24	657	13.253535	low
0	#04_#05	566	12.204376	low
0	#17_#18	349	12.052359	low
1	#11_#12	585	10.858193	low
1	#08_#09	50	8.884087	low
1	#07_#08	45	7.854191	low

APPENDIX C. Figures: Demonstration.

Figure 12. Scheme of how algorithm works.

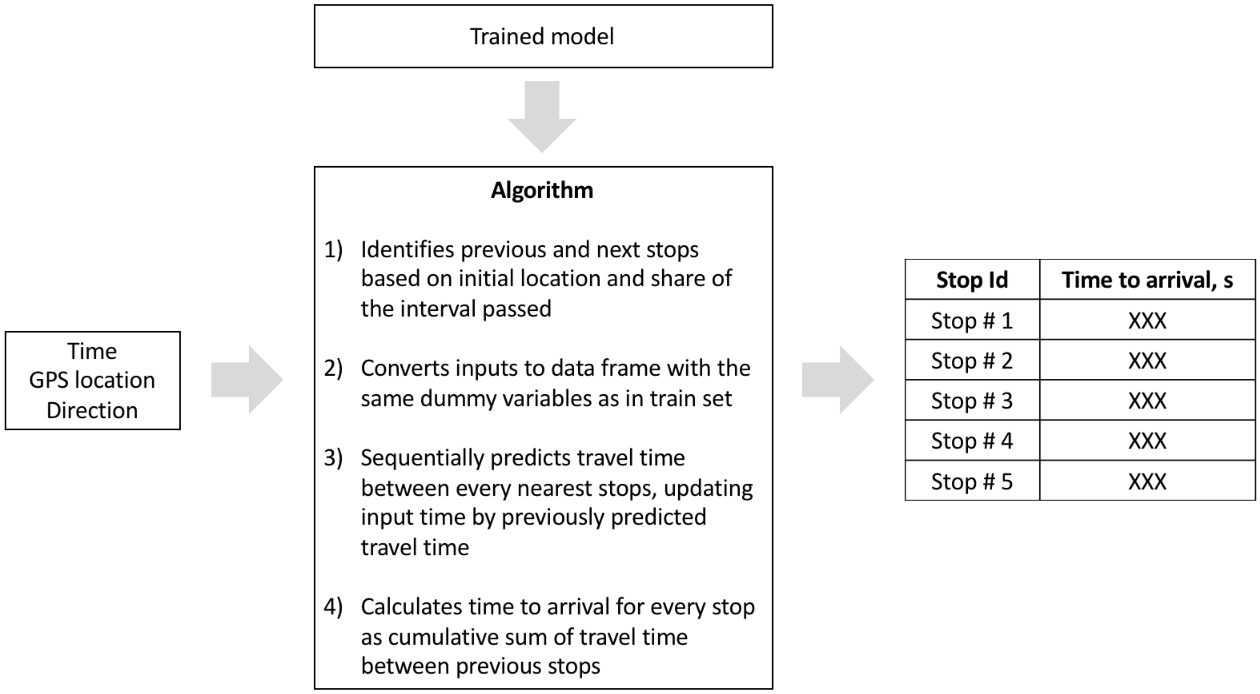
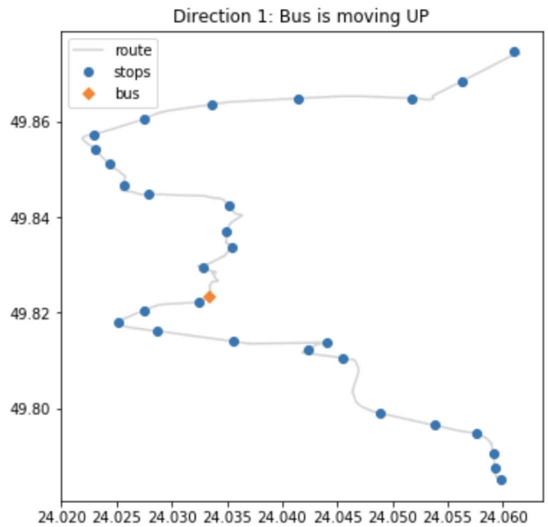


Figure 13. Demonstration.

```
Time: 2017-09-26T04:36:37.060401750
Direction: 1
Bus Location: [49.82356652125, 24.03342645875]
```

randomly generated bus location, time and direction (input)



visualization

previous and next stops for current bus location, share of interval passed

```
Bus is departed from stop #13: id=39164 and passed share=0.23 of the interval to stop #14: id=36559.

Next closest stops on the route: ['stop #14: id=36559' 'stop #15: id=36589' 'stop #16: id=36631'
 'stop #17: id=36685' 'stop #18: id=36711']
```

next 5 stops to be passed

	next_stop	seconds_to_arrival	arrival_time
0	stop #14: id=36559	155.847378	2017-09-26 04:39:12.907779750
1	stop #15: id=36589	298.304957	2017-09-26 04:41:35.365358750
2	stop #16: id=36631	433.488589	2017-09-26 04:43:50.548990750
3	stop #17: id=36685	628.974608	2017-09-26 04:47:06.035010750
4	stop #18: id=36711	795.359942	2017-09-26 04:49:52.420344750

output

APPENDIX C. Jupyter Notebooks attached.

1. ML Project Preprocessing.ipynb
2. ML Project Modeling.ipynb
3. ML Project Demonstration.ipynb