Taxi Trip Duration Prediction

Zijing Gu, Jingwei Li, Wuwei Lin
Carnegie Mellon University
Pittsburgh, PA 15213
{zijingg, jingwei5, wuweil}@andrew.cmu.edu

Abstract

In building modern intelligent transportation systems like taxi or ride-sharing apps, the accurate prediction of travel time is irreplaceable. It can not only improve the customers' experience on traveling, but also help taxi drivers manage their routes and orders in a more efficient way. This problem is challenging mainly due to its large dataset and the complex relationship between the model and features. By using the New York City's taxi record in 2017, this paper investigates several machine learning methods to predict the travel time. Since the number of the original feature is small, we introduce several external features that improve the performance of the models. Finally, we present evaluation results of the proposed method and ablation study on the design of the proposed model.

1 Introduction

Problem Formulation

Given the pick-up and drop-off location, our goal is to predict the travel duration based on the dataset X that contains the taxi travel records of the New York City. Other information may also be introduced to dataset X, e.g. weather, holiday and distance.

Brief Summary

Nowadays, ride-sharing apps are playing an increasingly important role in people's lives. It is especially important for customers and drivers to predict travel time in advance. Furthermore, the predicted travel time is also useful for city planning.

This problem is challenging because we need to find the appropriate features that have great impact on the travel time, which is not covered in the original dataset. Also, Since the dataset is very large, we need to explore some practical way to deal with it. In the following section, we describe our learning model and feature engineering method specifically.

In this paper, our main focus is on finding better learning model and useful external features. To extract valuable features from the raw dataset, the original data have been engineered to be new features in the learning model. We proposed several different models and introduced some new features. By comparing among all the models, we figure out our best model is the Gradient Boosted Tree. Also, the ablation test on features shows that the holiday has the most impact on our model.

The rest of this paper is organized as follows. In Section 2, we discuss background and related work. We introduce our data and make a visualization in Section 3. In Section 4, we discuss the method that has been used for solving this problem, which covers our model and feature engineering process. In Section 5, we evaluate our model and show the results. Finally, we make a conclusion in Section 6.

2 Background and Related Works

To predict the travel time, two different methods are frequently used. The first one is path-based method and the second one is origin-destination method. Since path-based method requires the whole route information of the travel, it is not applicable in our problem. So we used the origin-destination method, which is more relative to our raw dataset.

The main advantage of the origin-destination method is that it saves the time for calculating and estimating the travel path. Recently some researchers has used this method to predict travel time. Wang et al. (2016) introduced a nearest neighbor approach to the origin-destination method. In their approach, the travel time is predicted by calculating the average travel time of similar origin & destination location. Jindal et al. (2017) proposed a neural network to jointly estimate travel time and distance, but this method fails to capture spatial relationship between regions. Recently, Li et al. (2018) proposed a graph embedding learning method to capture the road links information. However, lacking the road links and trajectory information in the dataset, this method is not applicable to this task.

Compared to the path-based methods, the drawback of origin-destination methods is , instead of using the route information thoroughly, the origin-destination method ignores the underlying information about the road network. So the prediction result may not be very accurate.

3 Dataset

The dataset we used in this project is approximately 60 million taxi rides recorded by New York City taxicabs in 2017. The starting and ending locations are pre-processed by discretizing the region and reporting the index of the starting and ending sub-regions. Thus we do not have the exact pickup and drop-off latitudes and longitudes. The original dataset contains vendor ID (integer), passenger counts, pickup and drop-off date times (string), pickup and drop-off region ID (integer), and payment methods (integer). The dataset contains around 0.5% outliers, the duration of which is negative or larger than 8 hours. We dropped out these outliers in the dataset. We did not process beyond the given data or combine the given data with other datasets, but we did add more features which are elaborated in Section 4.1,

3.1 Data Visualization

To better understand the distribution of dataset, we planned to plot each record on an interactive map. We randomly selected 5% of the original data for the visualization. Since each record only has a regional location that is shared with many other records, plotting the frequency of records that fall into certain region makes more sense than directly plot each record. Thus we used the `ClusteredCircleViz` from the `mapboxgl` Python library and plotted the total counts of records in each region. Figure 1 and Figure 2 show the total number of observations for each region during the year of 2017. Figure 1 shows the pickup counts and Figure 2 shows the dropoff ones. From the graphs, each circle represents the frequency and its color and size depend on the value of the frequency. We see that most pickup and drop-off occurrences concentrate at midtown and downtown Manhattan, which makes sense because these are the busiest and most popular regions in NYC. We also found that the drop-off locations are more spread out to the east of New York (Brooklyn area). This is likely because the residential population is more concentrated in Manhattan area and people tend to call taxi to the more peripheral area for errands. When they go back home, since there are less time constraints, they tend to choose public transportation. Besides the Manhattan area, there are several hot spots that stand out, including John F. Kennedy International Airport and LaGuardia Airport (the two orange circles that stands out in Figure 1, 2), which is not surprising.

3.2 Coordinate Mapping

To be able to utilize the geographical feature, we decided to give each region its latitude and longitude so that we could calculate the distance between the given pickup and drop-off regions. We searched for the geojson file that contains the boundaries of each region and calculated the longitude and latitude of the centroid by taking average of each boundary to represent the actual location of each

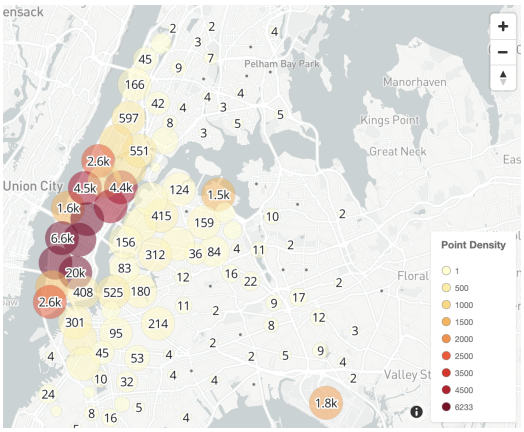

Figure 1: Pickup frequency of all regions in New York City

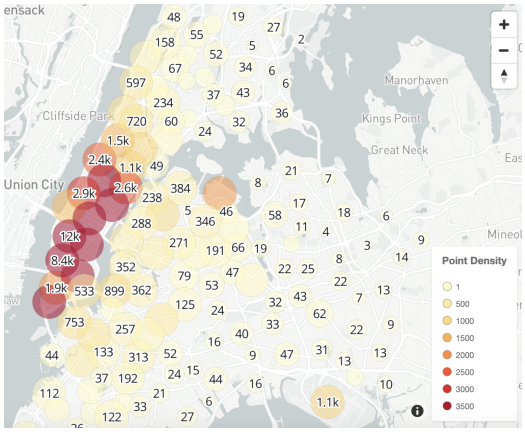

Figure 2: Drop-off frequency of all regions in New York City

region. The average of all the other coordinates is used for regions with the "unknown" region IDs. The original region IDs are kept as features.

4 Methods

4.1 Feature Engineering

We first appended the latitude and longitude of the centroid that represent each region that each record belongs to as features of the dataset. Then we parsed the date time string as a date time object and extracted year, month, day, weekday, hour and minute from the date and time of each ride as individual features. We calculated the duration of each ride by subtracting the drop-off time from pickup time, and dividing by 60 to obtain the trip duration in minute. We also added the log of the duration as an extra feature. We then used the `retrieve_hist_data` from `wwo_hist` Python library to obtain the historical weather from `WorldWeatherOnline` weather API for each observation including amount of rain amount and snow per hour in millimeters based on the pickup date and time. Besides, we also included US holidays as a binary feature based on the pickup date, where 1 represents that date is a holiday and 0 otherwise. Based on location coordinates, we calculated Haversine distance, Manhattan distances as well as driving directions. To utilize global information, we compute pickup and drop-off frequency of each region as an additional feature for each record based on which region they are located in, similar to what we showed in Section 3.1.

4.1.1 Feature Analysis

We split the dataset into training and testing sets with a ratio of 0.8 and 0.2 simply for the purpose of feature analysis. We plot the distribution of trip and logged duration as shown in Figure 3 and 4. As we can see, Figure 3 indicates a right skewed distribution while the logged duration is a normal distribution.

In Figure 5 and Figure 6, we showed that the training and testing sets have a roughly similar shape of distribution on total pickup as well as drop-off counts, i.e., a popular pickup location in training set is also a popular pickup location in test set, proving the randomness of data selection. By comparing both graphs, we can also infer that a popular pickup location is very likely a popular drop-off location. Figure 7 and Figure 8, on the other hands, show the total pickup and drop-off counts by date. Both graphs also show a similar pattern in train and data sets, as well as a high popularity correspondence between pickup dates and drop-off dates.

To better analyze the correlations between different features and duration, we focused on the duration less than 200 minutes. Figure 9, Figure 10, and Figure 11 show the correlation between duration and rain, snow, Haversine distance respectively. The Haversine distance is calculated according to equation (1), where r represents the average earth radius (6371 kilometers), φ are the latitudes and λ are the longitudes of two points along a great circle of sphere. From Figure 9 and Figure 10, we can

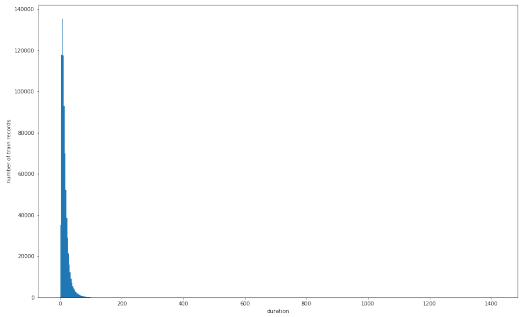

Figure 3: Distribution of duration in minutes

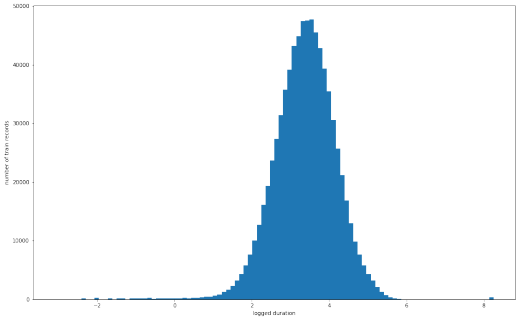

Figure 4: Distribution of log duration in minutes

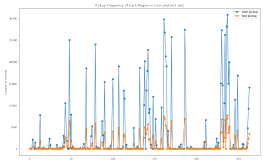

Figure 5: Total pickup counts of each region in train and test sets

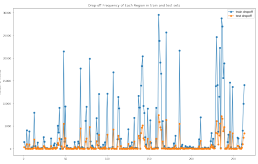

Figure 6: Total drop-off counts of each region in train and test sets

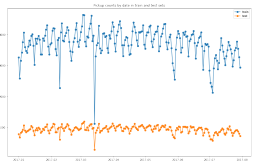

Figure 7: Total pickup counts by date in train and test sets

Figure 8: Total drop-off counts by date in train and test sets

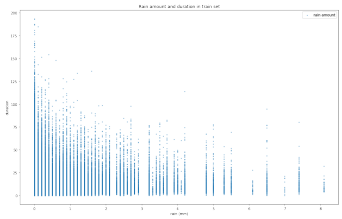

Figure 9: Correlation between amount of rain and duration less than 200 minutes

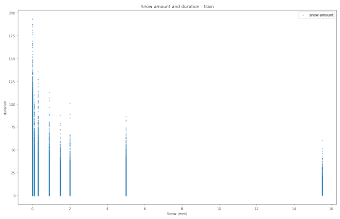

Figure 10: Correlation between amount of snow and duration less than 200 minutes

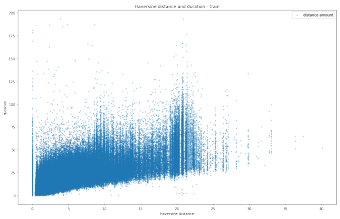

Figure 11: Correlation between Haversine and duration less than 200 minutes

see that larger amounts of rain and snow lead to shorter duration. This is possibly because during bad weather, people tend to not travel very far by taxi. Figure 11, on the other hand, shows a roughly positive correlation between distance and duration, which is reasonable.

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{1}$$

4.2 Encoding Categorical Features

The dataset contains both numerical features such as trip duration and region coordinates, and categorical features such as vendor id, date and time information. For continuous features, it is straightforward to represent them as a vector for each sample. However, the categorical features can be represented in multiple ways. We present three approaches in our models to represent categorical features.

One approach is to map categories to a set of 0-based continuous natural number set $\{0, 1, ..., N-1\}$, where N is the number of categories. In this way, each categorical feature can be represented with a single numerical column. The advantage of this approach is that categorical features can be represented in the same manner as continuous features. This makes the model design easier since we can apply general machine learning methods such as decision trees or regressions models to this

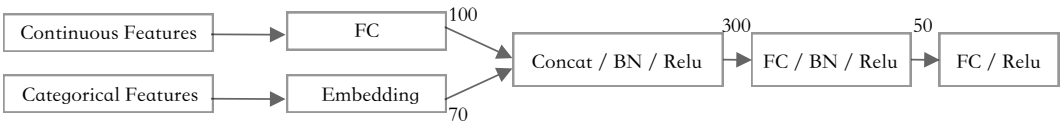

Figure 12: Network Architecture

task. However, since the categorical features do not necessarily have ordinal relationship, numerical representation might not be ideal.

Alternatively, we can represent the categorical features in one-hot encoding. The one-hot encoding is to represent a categorical feature with a total of d categories as a vector of d elements. Each element is a binary variable and represents a category. This representation alleviates the issue of the unexpected ordinal relationship in the numerical representation. However, if the number of the categories is large, it will make the model inefficient because the explosion of the number of parameters. For example, there are a total number of 265 regions, which need 265-d vectors to represent in one-hot encoding. Besides, this may also incur the curse of dimensionality (Bellman, 1966) and undermine the performance of the models.

We also explore learning-based encoding for categorical features using neural network. The idea is to maps category id to parametric vectors that can be optimized. We present this method in Section 4.5.

4.3 Gradient Boosted Tree

Decision trees have been proven practical in classification and regression tasks. We applied the gradient boosted tree model to the taxi trip duration prediction task. We use the implementation from XGBoost(Chen and Guestrin, 2016).

The main challenge in this task is to deal with huge dataset. After pre-processing, it takes about 12GB in memory, it is impossible to store the whole dataset in memory since the intermediate results during training can be $2 \sim 3$ times larger than the original dataset. Parallel training also requires the dataset being duplicated across different processes. To alleviate this problem, we use external memory mode of XGBoost, which caches the processed feature matrix in the disk. It should be noted that this method is scalable. It can be easily to extend to large distributed file systems such as HDFS if the dataset is even larger.

To accelerate training, we use an approximate method during tree splitting. When splitting the leaf node, instead of enumerating all possible values, only a small subset of candidates are sampled. These two strategies make the training efficient. It takes only 30 seconds for each iteration and the memory usage is less than 4GB.

Since decision-tree models are likely to over-fit, especially when the features of the dataset are not abundant, we restrict the maximum depth of each weak learner to 15.

4.4 Random Forest

The random forest model is another decision-tree-based meta estimator that fits a number of estimators. Similar to the gradient boosted tree model, without constraints of the tree depth, the model is likely to over-fit. We empirically choose the maximum tree depth to 15. We explore different number of estimators. The best model consists of 32 estimators. Due to the memory limit, we are unable to train larger random forest model.

4.5 Neural Network

We explore alternative methods to handle categorical features. Inspired by De Brébisson et al. (2015), we design a neural network with embedding layer for this task. The embedding layer can be seen as a simple lookup table that maps category id to parametric vector representation. Specifically, the embedding layer can be formulated as a matrix in $\mathcal{R}^{d \times k}$, where d is the number of categories and k is the embedded dimension. The embedding layer takes a category id as input and the id is then used as the column index. The selected column, which is a vector in $\mathcal{R}^k$, is the output of the embedding layer. The parameters of the embedding layer can be optimized with gradient descent. During the backward

Feature	Number of Categories	Embedding Dimension
Pick Up Region	265	20
Drop Off Region	265	20
Vendor ID	2	2
Payment Type	2	2
Quarter of the Day	96	13
Day of the Week	7	5
Week of the Year	53	8

Table 1: Categorical Embeddings for Neural Network

pass of the neural network, columns corresponding to category ids in the input are optimized. As a result, we are able to obtain the informative representation of the categorical features after training.

The network architecture is shown in Figure 12, output dimension of each layer is labels on the corner. The network takes two sets of features: continuous features, and categorical features. The continuous features are fed into a fully connected layer. Embedding layers are used for each categorical feature. The embedding dimension for categorical features are listed in Table 1.

As shown in Figure 12, after the embedding layers, the embedding vectors are concatenated into a 70-d vector. And then we concatenate it with the output of the fully connected layer, which is a 100-d vector representation for continuous features. This produces a 170-d vector. It is then fed into repeated blocks of fully connected, batch normalization (Ioffe and Szegedy, 2015) and ReLU (Glorot et al., 2011) layers. Finally the network outputs a scalar, which is the prediction for the taxi trip duration.

Since the duration in seconds can have a large range. The minimum duration in the dataset is only a few seconds, which the maximum can be a few hours. This makes it difficult to train the neural network, as models with large weight values is often unstable, meaning that it may suffer from poor performance during learning and sensitivity to outliers. Therefore, for the neural network model, we scale the trip duration to $[0, 1]$ range. The output of the network is scaled back to the original range to form the final prediction.

However, due to the extremely huge size of the dataset and the limited number of features, mini-batch based optimizers such as SGD do not perform well on this task. We provide further analysis in Section 5.

4.6 Model Ensemble

Model ensemble is a common technique to improve the performance of the classification and regression tasks. To aggregate predictions of different models, we train a regression model on top of the individual prediction of each model.

As mentioned in Section 4.5, mini-batch based optimizers do not perform well. This makes neural-network-based regression models impractical. Besides, since the dataset is huge, it is also computationally impossible to use kernel-based methods, which requires to compute NxN kernel matrix where N is the number of samples in the dataset. However, since the total number of the individual models in this work is small, it is easy and efficient to obtain the closed solution for linear regression. According to Eq.2, the closed solution for the linear regression model with squared error objective function can be efficiently computed in $O(NK^2)$ if $K \ll N$, where N is the number of samples in the dataset, K is the number of features for linear regression, which is the number of individual estimators in the setting of model ensemble.

$$\mathbf{w}^* = (X^T X)^{-1} X^T \mathbf{y} \tag{2}$$

Therefore, we choose a linear regression model with mean square error objective function for the mode ensemble. The output is taken as the final prediction.

5 Experiments and Results

In this section, we present experimental results of the taxi duration prediction on the NYC Taxi dataset. We first present the experiments and the results of our models. Then we present ablation study on a few design choices.

5.1 Evaluation Metrics

The dataset doesn't not provide training and testing split, therefore we report the average of the k-fold cross validation results on the validation set, where $k = 5$ in our experiment. To measure the prediction error, two metrics are used: mean absolute error (MAE) and root mean squared error (RMSE) in seconds as suggested by Wang et al. (2016). Specifically, for the prediction $\hat{y}$ and the true duration y in seconds, MAE and RMSE are defined as followed:

$$MAE = \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{N} \tag{3}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (\hat{y}_i - y_i)^2}{N}} \tag{4}$$

5.2 Results

A linear regression model is trained as baseline. We also report the result from Wang et al. (2016) as baseline, which simply computes the average of the neighbors. Here the neighbor is defined as the trips with pick up and drop off locations that are spatially closed.

We report the result of the models using the best hyper-parameters we selected on the training set. For the gradient boosted tree, we select numerical format to encode categorical features as described in Section 4.5. We train the model for 100 iterations since we observe the convergence of the model. Mean squared error is used as the objective function. For the random forest, we set the number of estimators to 32 using mean squared error as the objective function. Since in the random forest model, training of each individual estimator is independent. We train the model using parallely using multi-processing, which takes about 20 minutes for the whole training set on a AWS c5.4xlarge instance.

Table 2 shows the prediction error of each models. The decision-tree-based models perform best among all individual estimators. The best single estimator is the gradient boosted tree model. While the ensemble model provides the best final prediction results.

The performance of the neural network is not optimal. Since the dataset is huge, it takes 1.5 hours for a single epoch on a AWS c5.4xlarge instance. Due to the constraint of the computation resource, we train only up to 20 epochs. We also train a one-layer neural network without activation layers, which is equivalent with the unregularized linear regression model. In this case, the neural network is convex, which is expected to converge to the global optimum. Surprisingly, with SGD or other optimizers in the recent literature such as RMSProp (Tieleman and Hinton, 2012) and Adam (Kingma and Ba, 2014), the result is worse than the closed solution of the linear regression. We suspect that the dataset is too huge and mini-batch based methods are difficult to converge.

Model	MAE	RMSE
Linear Regression	431.12	632.87
Wang et al. (2016)	354.27	504.16
Random Forest	260.74	400.39
Gradient Boosted Tree	202.29	339.63
Neural Network	623.35	827.28
Ensemble	202.24	339.56

Table 2: Prediction Error of Different Models

Features	MAE	RMSE
Original	203.03	340.01
+ Coordinates & Distance	203.03	340.00
+ Holiday	202.59	339.64
+ Weather	203.01	339.99
+ Region Frequency	202.56	339.71
+ All External Features	202.29	339.63

Table 3: Effects of External Engineering

5.3 Ablation Study

5.3.1 Feature Engineering

To verify the effects of the external features, we train the best model in Table 2, the gradient boosted tree model, with different sets of features. Table 3 shows than compared with original features, adding external features boost model performance.

"Original" is the evaluation of the original dataset without any external features. Based on the original dataset, we add different external features to it the evaluate the performance. In "+ Coordinates & Distance", we mapped the region IDs to longitudes and latitudes, while keeping the original ids as categorical features. In "+ Holiday", we add the holiday information, which is a single column of binary value to the original dataset. "+ Weather" shows the result of the original dataset plus snowfall and precipitation data. In "+ Region Frequency", we add the pick up and drop off frequency calculated on the whole training set to each sample.

Experiment results show that region frequency is most helpful. This is likely because that this feature provides global information of the whole dataset. The last row in Table 3, "+ All External Features", is the original dataset plus all external features aforementioned, with which we obtain the performance superior than the original dataset and the dataset with any single external features.

We also plot the importance score (Chen and Guestrin, 2016) of features in the gradient boosted model as shown in Figure 13. The result is similar to the observation in Table 3.

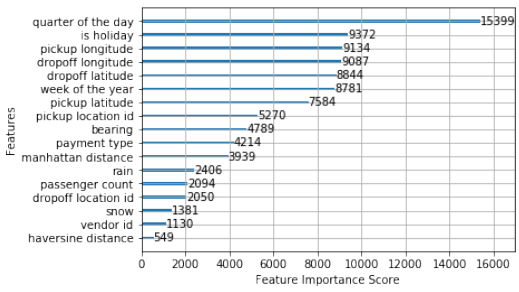

Figure 13: Feature Importance of the Gradient Boosted Tree

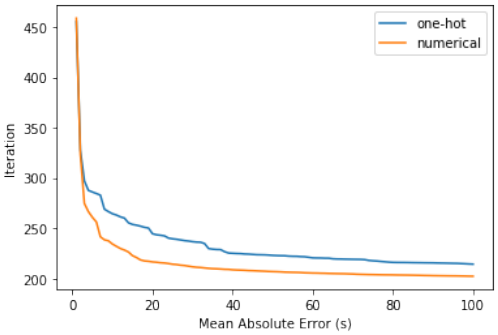

Figure 14: MAE on test set for different encoding methods

5.3.2 Encoding of Categorical Features

To compare the performance of different encoding methods for categorical features described in Section 4.2, we train the gradient boosted tree model with different encoding methods. Figure 14 shows MAE on the test set after each iteration. The result suggests that both encoding methods yield similar performance after a sufficient number of iterations. However, using one-hot encoding makes the model slower to converge because of the increase of the dimension.

6 Conclusion and Future Work

In summary, we explored different learning methods and various external features. The result shows that the Gradient Boosted Tree method has the best performance, while the holiday feature and the region frequency feature are the most important external features. Considering the poor performance of the neural network, we intend to include more features to improve its result. Besides, we only used the origin-destination method. In the next step, we will include road network information to estimate the travel path. Overall, our future work will include engineering more external features and using graphical models to utilize road link information.

References

Hongjian Wang, Yu-Hsuan Kuo, Daniel Kifer, and Zhenhui Li. A simple baseline for travel time estimation using large-scale trip data. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPACIAL '16, pages 61:1–61:4, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4589-7. doi: 10.1145/2996913.2996943. URL http://doi.acm.org/10.1145/2996913.2996943.

Ishan Jindal, Tony, Qin, Xuewen Chen, Matthew Nokleby, and Jieping Ye. A unified neural network approach for estimating travel time and distance for a taxi trip, 2017.

Yaguang Li, Kun Fu, Zheng Wang, Cyrus Shahabi, Jieping Ye, and Yan Liu. Multi-task representation learning for travel time estimation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 1695–1704, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5552-0. doi: 10.1145/3219819.3220033. URL http://doi.acm.org/10.1145/3219819.3220033.

Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966. ISSN 0036-8075. doi: 10.1126/science.153.3731.34. URL https://science.sciencemag.org/content/153/3731/34.

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. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL http://doi.acm.org/10.1145/2939672.2939785.

Alexandre De Brébisson, Étienne Simon, Alex Auvolat, Pascal Vincent, and Yoshua Bengio. Artificial neural networks applied to taxi destination prediction. In *Proceedings of the 2015th International Conference on ECML PKDD Discovery Challenge - Volume 1526*, ECMLPKDDDC'15, pages 40–51, Aachen, Germany, Germany, 2015. CEUR-WS.org. URL http://dl.acm.org/citation.cfm?id=3056172.3056178.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *JMLR Proceedings*, pages 448–456. JMLR.org, 2015.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *AISTATS*, volume 15 of *JMLR Proceedings*, pages 315–323. JMLR.org, 2011. URL http://dblp.uni-trier.de/db/journals/jmlr/jmlrp15.html#GlorotBB11.

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.