A Graph-Based Machine Learning Approach to Realistic Traffic Volume Generation

by

Kyle Otstot

An Undergraduate Thesis Submitted in Partial Fulfillment of the Requirements for

Barrett, the Honors College at Arizona State University

Defense Date: Wednesday April 27th, 2022

Director: Dr. Gennaro De Luca

Polytechnic Sch IT Prgms

Second Reader: Dr. Yinong Chen

School of Computing and Augmented Intelligence

ABSTRACT

In this work, we explore the potential for realistic and accurate generation of hourly traffic volume with machine learning (ML), using the ground-truth data of Manhattan road segments collected by the New York State Department of Transportation (NYSDOT). Specifically, we address the following question– *can we develop a ML algorithm that generalizes the existing NYSDOT data to all road segments in Manhattan?*– by introducing a supervised learning task of multi-output regression, where ML algorithms use road segment attributes to predict hourly traffic volume. We consider four ML algorithms– K-Nearest Neighbors, Decision Tree, Random Forest, and Neural Network– and hyperparameter tune by evaluating the performances of each algorithm with 10-fold cross validation. Ultimately, we conclude that neural networks are the best-performing models and require the least amount of testing time. Lastly, we provide insight into the quantification of "trustworthiness" in a model, followed by brief discussions on interpreting model performance, suggesting potential project improvements, and identifying the biggest takeaways. Overall, we hope our work can serve as an effective baseline for realistic traffic volume generation, and open new directions in the processes of supervised dataset generation and ML algorithm design.

ACKNOWLEDGMENTS

This work is an extension of the ASU CSE Capstone Project. Specifically, this work is the author's contribution to his team project, namely "Shortest Path in Dynamic Traffic Simulation", supervised by Dr. Yinong Chen and Dr. Gennaro De Luca. The team members include Aleczander Allen, Kyle Otstot, Lucy Song, Hai (Mark) Truong, and Zakyre Vanstrom.

TABLE OF CONTENTS

1. INTRODUCTION

In this project, we dissect the notion of *shortest-route computation*, a widespread application of graph-based algorithms and a fundamental component of popular navigation interfaces. Specifically, we evaluate the efficacy of dynamic shortest path (DSP) algorithms by focusing on the collection and approximation of traffic volume, an important parameter in DSP algorithms. In doing so, we explore the practical means of hourly traffic volume collection, and identify the road network of Manhattan, New York City as a prime candidate for further investigation. The New York State Department of Transportation (NYSDOT) reports hourly traffic volume counts, but only for a small subset of Manhattan's road segments, so we seek to develop an algorithm that approximates the hourly volume for unreported road segments. For this, we turn to machine learning (ML) and pose the following question: can we develop a ML algorithm that generalizes the existing NYSDOT data to all road segments in Manhattan?

Addressing the question, we introduce a supervised learning task of multi-output regression, where road segment attributes are used by the ML algorithms to predict 24-hour traffic volume vectors. Inputs take the form of 4 OpenStreetMap (OSM) attributes and 8 engineered attributes preprocessed and encoded into feature vectors, while outputs are simply composed of hourly traffic volume taken from the NYSDOT database. We consider four ML algorithms– K-Nearest Neighbors, Decision Tree, Random Forest, and Neural Network– and hyperparameter tune by evaluating the mean absolute error (MAE)– a robust and interpretable performance metric– with 10-fold cross validation for each setting. The optimal results are reported and compared between the algorithms, and traffic volume predictions given by each algorithm are illustrated. Overall, we find that the neural network algorithm performs the best and requires the least amount of testing time. Lastly, we give a thorough evaluation of our project

as a whole, including discussions on model trustworthiness, model performance, project improvements, and project merits. We hope our work can serve as an effective baseline for realistic traffic volume generation, and open new directions in the processes of (1) generating a supervised learning dataset, and (2) designing well-performing ML regression algorithms.

Summary of our contributions:

- ★ We synthesize the NYSDOT and OSM databases into a supervised learning dataset under the task of multi-output regression, aiming to find a relationship between road segment information and hourly traffic volume in Manhattan, New York City. The input features are derived from preprocessed road segment attributes, while the output targets are the corresponding 24-hour volume vectors.
- ★ We consider four standard ML algorithms– K-Nearest Neighbors, Decision Tree, Random Forest, and Neural Network– for solving the aforementioned task of traffic volume prediction. In doing so, we hyperparameter tune with 10-fold cross validation, report the top results, and illustrate the network-wide predictions for each algorithm.
- ★ We give a thorough evaluation of the project results, including a hypothesis test on train and test distribution similarity to assess model trustworthiness, discussion on interpreting model performance, identification of potential avenues to improve upon in future work, and reflection of the project's positive takeaways.

2. PROJECT MOTIVATION & PROBLEM SETUP

As the capabilities of mobile technology continue to expand, navigation applications (*e.g.,* Google Maps, Apple Maps) are becoming more relevant in daily routines ("How Many People Use Google Maps Compared to Apple Maps? July 2021"). A fundamental use-case of

navigation is the *giving of directions*: when a vehicle is located at point A and wants to reach point B, the application will compute a route that leads the vehicle from A to B in the shortest amount of time possible. However, the defining characteristic of navigation applications (compared to a static map) is the implementation of a *dynamic shortest path algorithm*, which acknowledges that the shortest path between points A and B is not fixed, but rather a function of time. For instance, the shortest path between two intersections at 7:00am– when people commonly travel to work– may be different than the one at 5:00pm– the midst of rush hour; Figure 1(a) illustrates a simple example of this phenomenon.

Specifically, a dynamic shortest path algorithm requires two pieces of information: the road network structure and hourly edge weights. A *road network* is a graphical representation of a real-world road map, defined by nodes (representing intersections) and directed edges (representing road segments). Road networks typically contain additional information, including node locations (GPS coordinates), road names, speed limits, and number of lanes. Once the network structure is established, the dynamic shortest path algorithm will apply *hourly weights* to the existing directed edges (road segments); these weights should represent the expected time to travel the road segment, at a given hour in the day. Once the algorithm has access to this information, it can dynamically compute a shortest path algorithm (*e.g.,* Djikstra's) for each hour, given a starting and ending location in the network.

In this project, we consider the development of a dynamic shortest path algorithm, given a *real-world example* of a road network and hourly edge weights. Before applying hourly edge weights to the road network, it is necessary to understand how real-world traffic is measured. Specifically, traffic is primarily quantified by two metrics: traffic volume and traffic density. *Traffic volume* is the number of passing cars per unit of time (cars / hour), while *traffic density* is

the number of cars in a given segment of the road (cars / mile). Combining these two metrics, along with the length of a road segment, yields the overall formula for *travel time*, the metric desired for the hourly edge weights:

$$travel\ time\ (hours) = \frac{density\ (cars/mile) \times road\ length\ (miles)}{volume\ (cars/hour)}$$

Therefore, the computation of the hourly edge weights requires (1) hourly traffic volume data, (2) hourly traffic density data, and (3) road segment lengths. Since (3) is static, this can be easily obtained from the road network structure. Although access to (1) is relatively scarce, we found that the *New York State Department of Transportation* (NYSDOT) reports hourly traffic volume counts for a fraction of road segments in *Manhattan, New York City* ("HDSB Raw Traffic Data"). Lastly, (2) is not easily recorded, so we did not find any public data for traffic density; therefore, we set traffic density to be constant, and focus on strictly working with traffic volume counts.

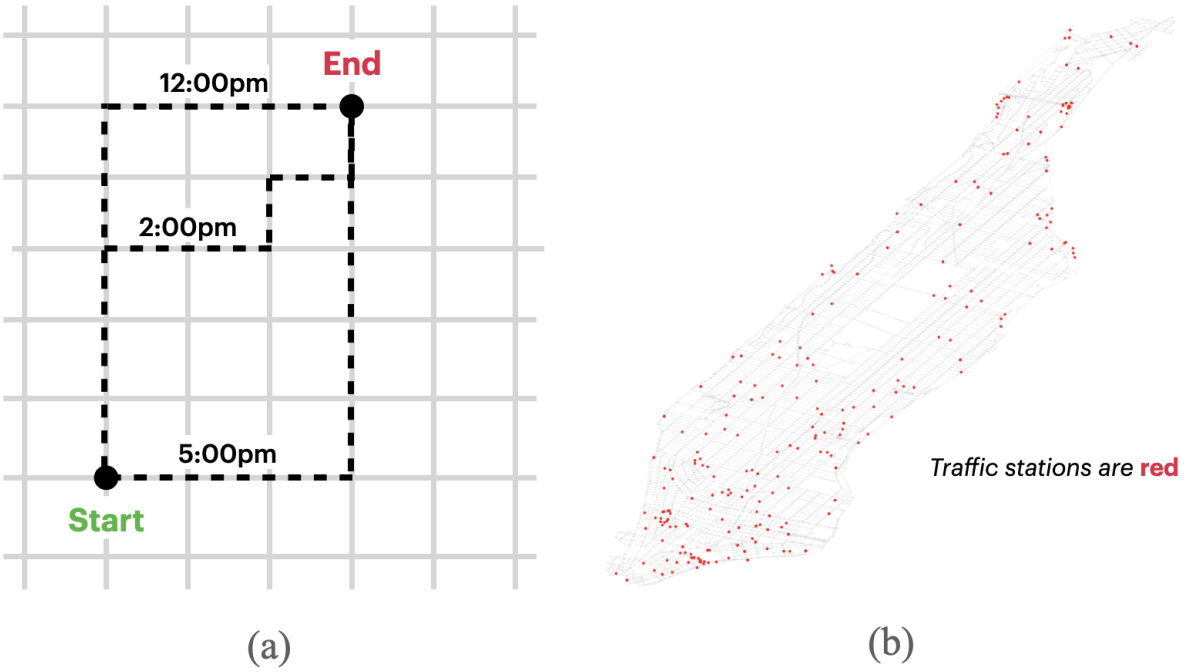


Figure 1: (a) an illustration of dynamic shortest paths, (b) traffic station placement in Manhattan.

Since we found data to help compute hourly edge weights for Manhattan road segments, we must obtain the full road network structure of Manhattan for the dynamic shortest path algorithm. In doing so, we use *OpenStreetMap* (OSM), a free geographic database of the world; specifically, we extracted a GraphML file of Manhattan from the OSM database ("Basic Network Analysis Tutorial"). In this file, a directed graph of Manhattan exists with the following data: nodes (intersections), node data (*e.g.,* GPS coordinates, roads), directed edges (road segments), and edge data (*e.g.,* road name, length, speed limit). Therefore, the GraphML file is sufficient in representing the road network structure of Manhattan, and will be used by the dynamic shortest path algorithm.

Although we have access to some hourly traffic volume, the problem of insufficient data must be addressed. In order to make predictions with a dynamic shortest-path algorithm, we need hourly traffic volume for *all* ~9,600 road segments (edges) in the network. However, NYSDOT only receives hourly traffic volume from a *small subset* of road segments; specifically, the hourly traffic volume is collected by 310 *traffic stations*, each tracking the number of passing vehicles per hour for their respective road segments. Figure 1(b) plots the 310 traffic stations against the entire Manhattan road network.

Addressing this problem, we pose the following research question: **can we develop a machine learning (ML) algorithm that generalizes the existing NYSDOT data to all road segments in Manhattan?** That is, can we train a ML model to make accurate predictions on ~9,600 unforeseen road segments by learning the traffic patterns from 310 recorded examples? To answer this question, we perform two main tasks. First, we generate and preprocess a *supervised learning* dataset that contains road segment attributes as the input and hourly traffic volume counts as the output; this step is further described in Section 3. Next, we look to evaluate

the performance of several ML algorithms on the generated dataset, which is further described in Section 4. If a ML model can accurately predict traffic volume for all road segments, then we have data– in the form of hourly edge weights– for the hypothetical dynamic shortest path algorithm. Moving forward, this project is strictly concerned with the development of ML models that predict hourly traffic volume.

3. DATASET GENERATION & PREPROCESSING

In this section, we detail the generation process of a supervised learning dataset, where the input features are derived from preprocessed and encoded road segment attributes, and the output targets are simply composed of the corresponding 24-hour volume vectors. We provide an overview of traffic stations, followed by a description of the implemented traffic station-to-road segment mapping algorithm, analysis of OSM-based and engineered road segment attributes, and brief discussion on K-fold cross validation.

3.1. Traffic Station Overview

As previously stated, a *traffic station* is a device that counts the number of passing vehicles and reports the hourly traffic volume. There exist a total of 310 traffic stations reported in Manhattan. Specifically, each traffic station belongs to one road segment (directed edge in the road network) and contains a 24-hour vector of traffic volume, reported by NYSDOT. Traffic stations are important because they give us the “ground-truth” output for our ML algorithms, but the stations themselves are not our desired input. Rather, we are looking to learn a function that receives any road segment in Manhattan, and predicts a 24-hour volume vector. To do this, we must train a ML model on pairs of road segment attributes (input) and 24-hour traffic volume

(output). After training, we test on pairs *not* used for training to determine how accurate the model is at predicting traffic volume. Thus, for our dataset, we need pairs of road segments and 24-hour volume vectors; this can be done by mapping each traffic station to the road segment containing it. Once the mappings are complete, we can determine the necessary attributes from the road segments containing traffic stations– we call these stationed road segments (SRS's).

3.2. Mapping Traffic Stations to Road Segments

Finding the road segment that contains a given traffic station turns out to be a nontrivial task, primarily because the traffic stations are defined in the NYSDOT database, while road segments are defined in the OSM road network. Therefore, the task comes down to figuring out a way to use the traffic station information in the NYSDOT database to determine the corresponding road segment ID in the OSM network. To do this, it is important to understand the structure of the NYSDOT database, which is presented in Figure 2.

	RCSTA	Roadway Name	Station Start	Station End	Federal Direction	Latitude	Longitude
0	44080	E 13TH ST	AVENUE C	AVENUE D	Eastbound	40.72735	-73.97518
1	44080	E 13TH ST	AVENUE C	AVENUE D	Westbound	40.72735	-73.97518
2	44080	E 13TH ST	AVENUE C	AVENUE D	Combined Total	40.72735	-73.97518
3	44130	E 8TH ST	FIFTH AVE	3RD AVENUE	Eastbound	40.73031	-73.99174
4	44130	E 8TH ST	FIFTH AVE	3RD AVENUE	Combined Total	40.73031	-73.99174
...	...	...	...	...	...	...	...
324	14250	SOUND VIEW AVE	ROSEDALE AVE	STORY AVE	Southbound	40.82272	-73.86753
325	14250	SOUND VIEW AVE	ROSEDALE AVE	STORY AVE	Combined Total	40.82272	-73.86753
326	54250	74TH ST	GRAND AVE	CALAMUS AVE	Northbound	40.73180	-73.88825
327	54250	74TH ST	GRAND AVE	CALAMUS AVE	Southbound	40.73180	-73.88825
328	54250	74TH ST	GRAND AVE	CALAMUS AVE	Combined Total	40.73180	-73.88825

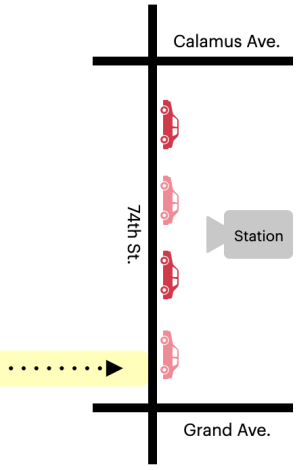


Figure 2: Traffic station information taken from the NYSDOT public database.

The “RCSTA” column combined with the “Federal Direction” column represents the station ID. In the database, every station contains the following information: GPS coordinates (“Longitude” + “Latitude”), the road parallel to the station (“Roadway name”), and the two perpendicular roads defining the endpoints of road segment (“Station Start” + “Station End”). This information is also illustrated in Figure 2. On the other hand, the OSM database simply defines a road segment by a start intersection ID and end intersection ID. In addition, each road segment contains a road name, and each intersection contains a GPS coordinate.

One strategy to map a traffic station in NYSDOT to a road segment in OSM is by using *road names*. We begin with a given traffic station: from here, we can find the road names making up the start intersection (“Roadway name” + “Station start”) and end intersection (“Roadway name” + “Station end”). After making a function that finds the OSM intersection ID given two intersecting road names, it is easy to use the start (end) intersection road names to find the start (end) intersection ID. Once we have these two IDs, we can look up the road segment in the OSM network, which gives us our desired result. This approach is fairly simple except for the fact that it relies on one very important assumption– that the road names in NYSDOT are consistent with those in OSM. Unfortunately, this assumption is not always true, and one conflicting example is shown in Table 1.

Database	Road name
NYSDOT	E 79TH ST
OSM	East 79th Street

Table 1: An example of how the same road can be named differently by NYSDOT and OSM.

From this example, it makes clear that some intuitive token conversions are possible: changing {N, S, E, W} to {North, South, East, West}, uppercase to title (only capitalize the first word), {St, Ave, Rd} to {Street, Avenue, Road}, and so on. However, even with these conversions, there are still seemingly-unresolvable inconsistencies. Therefore, we need another strategy that is not reliant on road names; this leads us to a mathematical approach that makes use of GPS coordinates. The algorithm goes as follows: from a given traffic station, we can find the GPS coordinate ("Longitude" + "Latitude"), namely (x_s, y_s), from the NYSDOT database. Since edges represent road segments in the OSM network, we are looking for the similarly-directed road segment (edge*) that satisfies the following formula:

$$\text{edge}^* = \min_{E \in \text{edges}} d^{(E)} := \left| y_s - \left(m^{(E)}(x_s - x_1^{(E)}) + y_1^{(E)} \right) \right|,$$

where $(x_1^{(E)}, y_1^{(E)})$ and $(x_2^{(E)}, y_2^{(E)})$ are the GPS coordinates of edge E's start and end intersections, respectively. The slope of each edge E is computed with:

$$m^{(E)} = \frac{y_2^{(E)} - y_1^{(E)}}{x_2^{(E)} - x_1^{(E)} + \epsilon}$$

where ε is a very small amount of random noise to ensure with high probability that the denominator is nonzero. Figure 3(a) illustrates a simple example of this algorithm.

This approach relies on the basic assumption that road segments are perfectly straight. Although there are few exceptions, we believe this assumption generally holds in almost all cases, so the approach effectively creates the mappings we desire. Figure 3(b) visualizes all the traffic station + road segment pairings determined by the algorithm; moving forward, we use these pairings.

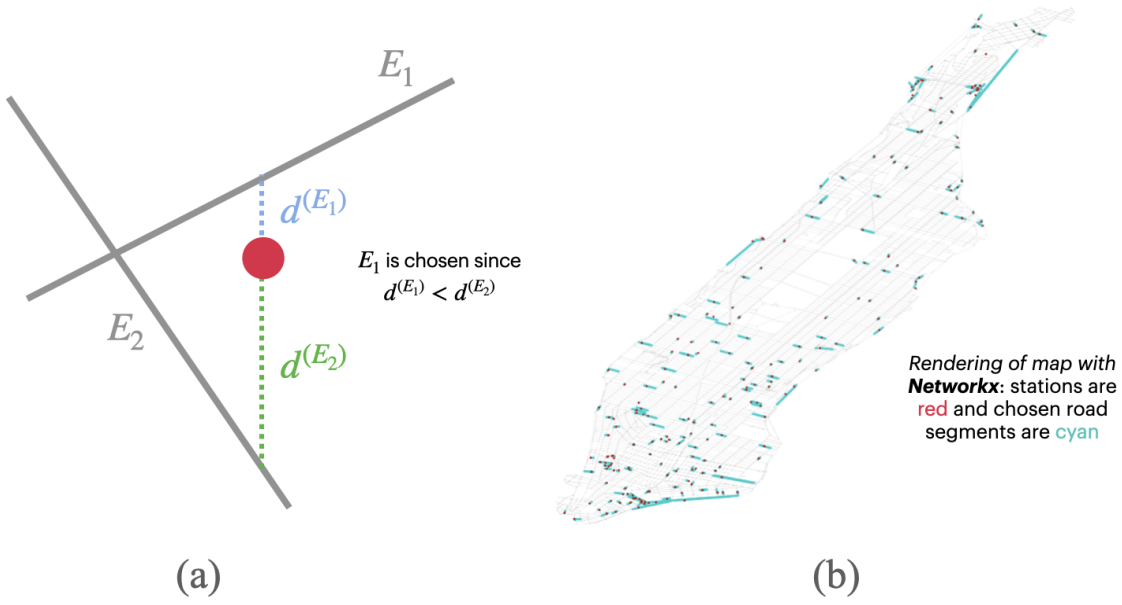


Figure 3: (a) An example of the mapping algorithm, (b) The finalized mappings in Manhattan.

3.3. Dataset Generation Pipeline

Now that we have established a way to make pairings of traffic stations and road segments, we can begin discussing the generation of features and targets for machine learning. We refer to the full process as the *dataset generation pipeline*, and a diagram can be found in Figure 4(a). First, we begin with a pool of traffic station + road segment pairings. For each pairing, we use the road segment to extract and engineer attributes from the OSM network (see subsections IV and V), and we use the traffic station to extract the 24-hour volume vector from the NYSDOT database. Therefore, every example contains a list of attributes (input) and a 24-hour volume vector (output). Section 4 will seek to answer the following question: given the *attributes*, how well can we predict the *24-hour volume*?

To answer this, we must "encode" the list of attributes into a vector of numbers that ML models can interpret; these encoded vectors are called *features*. Since the 24-hour volume

vectors are already in proper form, we can refer to these as *targets*. Every SRS (station + road segment) contains a feature and a target. In Figure 4(b), the features and targets are organized by SRS into two matrices. We do this because many ML algorithms (*e.g.,* decision trees, random forests, neural networks) are implemented to learn on feature and target matrices.

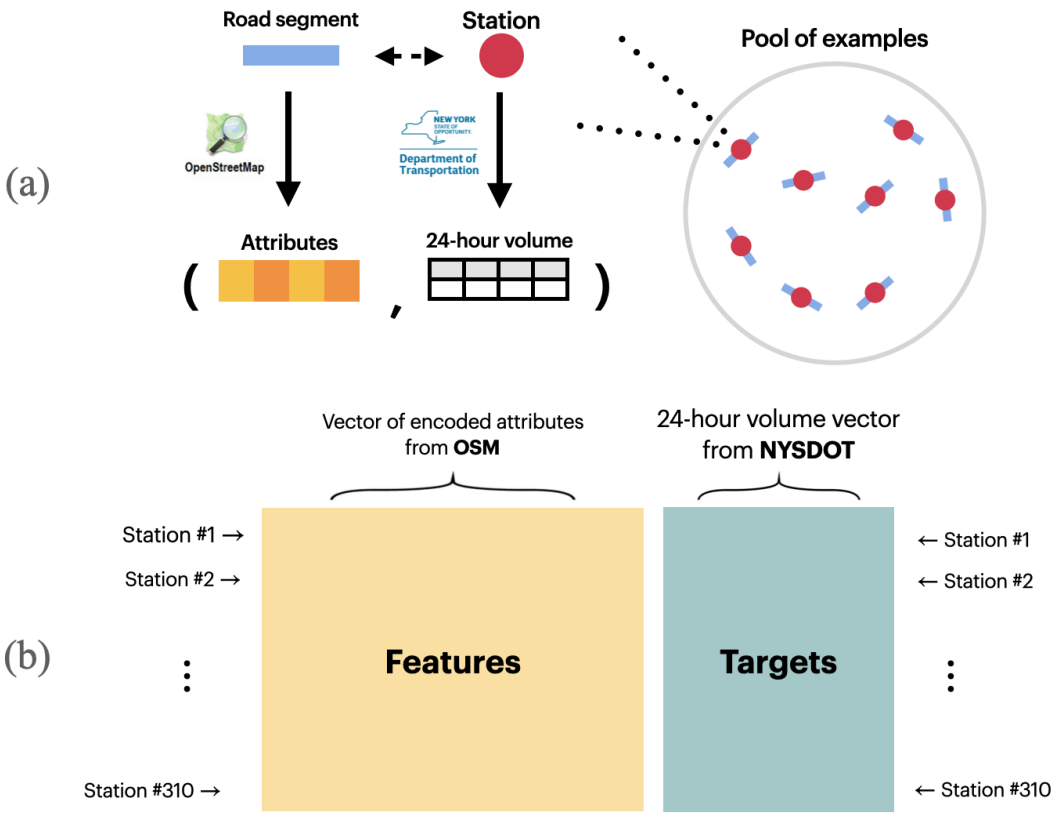


Figure 4: (a) The dataset generation pipeline, (b) An illustration of the feature and target matrices.

3.4. OSM Feature Selection

The next two subsections deal with the selection and engineering of attributes from a given road segment. All the attributes discussed below can be found in the OSM road network. However, identifying attributes is not enough; for ML success, we are particularly interested in the predictive capability of each attribute for *traffic volume*. Typically, this is done by plotting

traffic volume against the attribute values, then analyzing the trends; however, the traffic volume space is 24 dimensions, so it is impossible to visualize without dimensionality reduction. We solve this problem by reducing the 24-hour volume vector down to a *single value* representing the entire vector, so that we can plot a 2D graph of attribute value vs. traffic volume. Specifically, we claim that the *daily volume count* (*a.k.a.,* vector-wise sum) is a very useful single-value representation of the 24-hour traffic volume. Evidence for this claim is twofold– normalization, and principal component analysis (PCA).

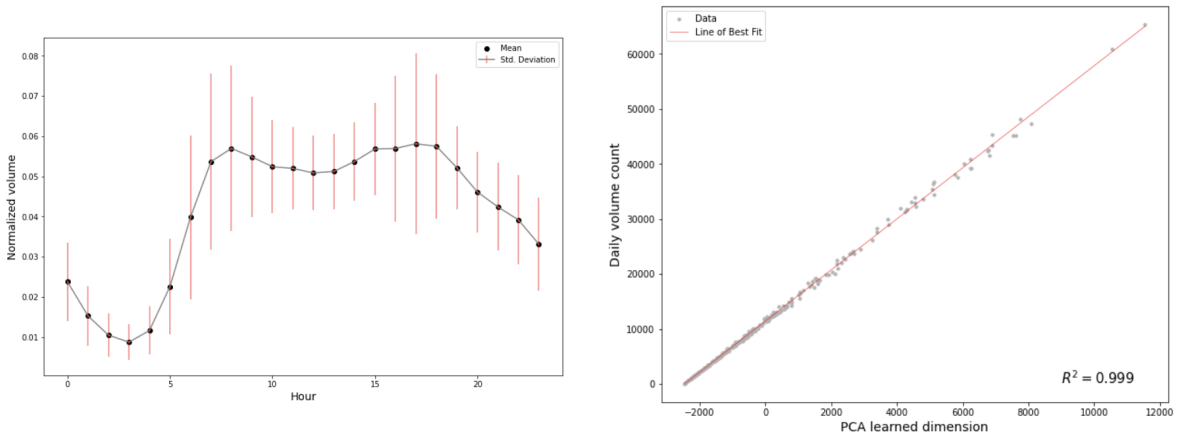


Figure 5: (a) Normalization across hourly traffic volume, (b) PCA single dimension vs. daily volume.

For normalization, if we divide each 24-hour volume vector by its sum, then the hour-wise distributions are very compact and reflect a clear pattern in daily traffic, as shown in Figure 5(a). For example: 12am-5am is a period of low volume because most people are sleeping; 7am-9am reflects high volume because people typically commute to work; and 4pm-6pm also experiences high volume because people return home from work (*a.k.a.,* rush hour). For principal component analysis, if we use Scikit-Learn's implementation of PCA

("sklearn.decomposition.PCA — scikit-learn 1.1.1 documentation") to project the 24-dimension traffic volume space onto a single dimension that best preserves variance among the samples, we observe in Figure 5(b) that the learned single-dimension distribution of values is almost perfectly ($R^2 \approx 1$) directly related to daily volume count. As a result, we will commonly appeal to daily volume count for attribute analysis. There are four OSM attributes identified below: GPS coordinate, road length, speed limit, and number of lanes. In the following paragraphs, we will provide insight into each attribute, as well as discuss the preprocessing and encoding methods we use to create ML-friendly features.

GPS Coordinate: In the OSM data, a GPS coordinate (longitude x & latitude y) is reported for each intersection. Since road segments are defined by two intersections, we simply take the mean of the two coordinates– $(x_{seg}, y_{seg}) = \left(\frac{1}{2}(x_{int1} + x_{int2}), \frac{1}{2}(y_{int1} + y_{int2})\right)$ – to find a centerpoint of the road segment. Figure 6(a) plots the GPS coordinates for each SRS and colors them by levels of daily traffic volume. In general, we suspect that road segments within a geographic cluster are likely to reflect similar patterns in traffic volume; for example, residential zones may be busier in the morning or evening, while nightlife hotspots may be busier at night. In the figure, we observe that a non-linear relationship exists between longitude-latitude and traffic volume; small geographic clusters appear to share some traffic volume similarity, so if we feed this data to a large-capacity model, then some patterns may emerge.

Since the relationship between GPS coordinate and traffic volume appears to be rather complex, we preprocess the 2D coordinates via *K-Means clustering*. This algorithm finds the K centerpoints (*a.k.a.* centroids) that best represent the distribution of data. Each centroid defines a cluster, and road segments are assigned to closest centroid (and labeled $1 \ldots K$). Specifically, we find that $K = 20$ allows the dataset to preserve local cluster information while encouraging

cluster diversity. Using Scikit-Learn's KMeans implementation ("sklearn.cluster.KMeans — scikit-learn 1.1.1 documentation"), we encode each coordinate with a vector $v = (v_1, v_2, ..., v_k)$ where v_i is 1 if the segment belongs to cluster i, or 0 otherwise. Figure 6(b) gives an example of K-Means clustering on the SRS coordinates for $K = 10$.

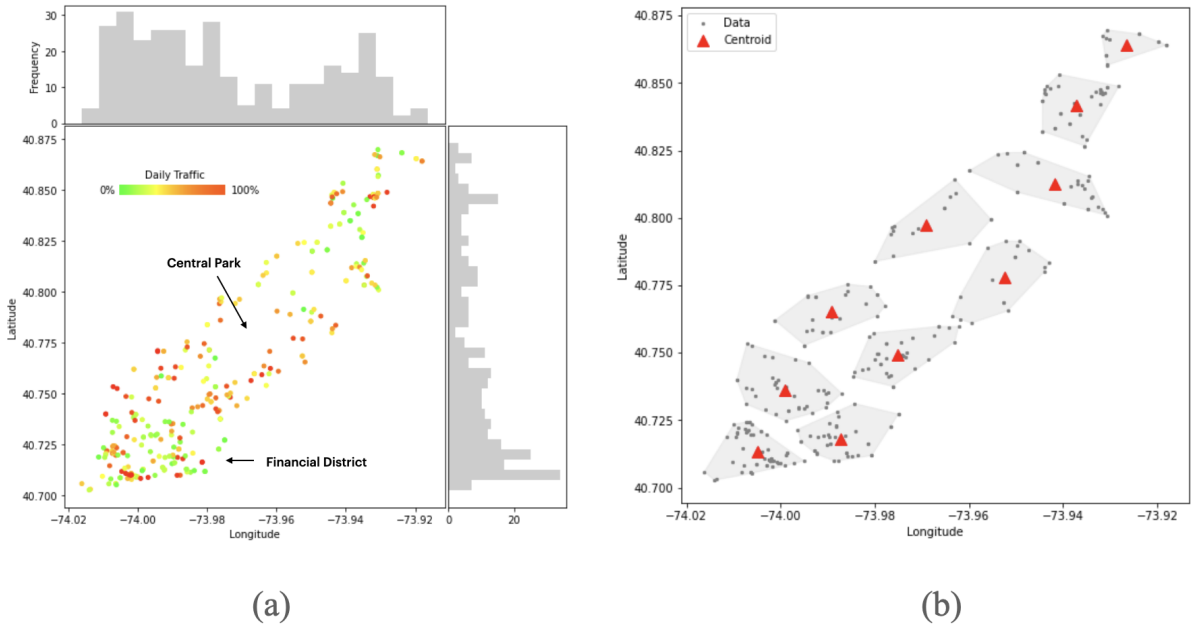


Figure 6: (a) Color-coded GPS coordinates for each traffic station, (b) Example of 10-means clustering.

Road Length: This OSM attribute defines how long a road segment is, in meters. We suspect that longer road segments are more important to the structure of the network, thus possessing higher traffic volume levels. Figure 7(a) plots the daily volume counts of each SRS against road length, and provides a line of best fit. Due to the very low coefficient of determination (R^2), we conclude that virtually no linear relationship exists between road length and traffic volume. However, we note that outliers (the longest road segments) tend to reflect higher volume counts. It appears that road length (in isolation) is a poor predictor of traffic

volume, but this important attribute may still serve use in a high-capacity regressor where complex relationships between attributes and volume counts are exposed.

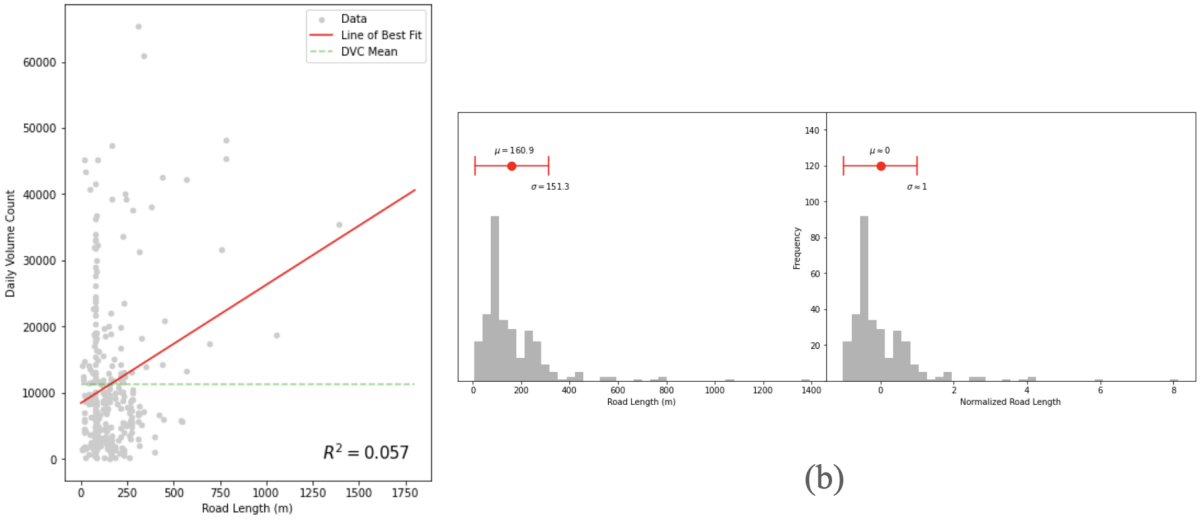
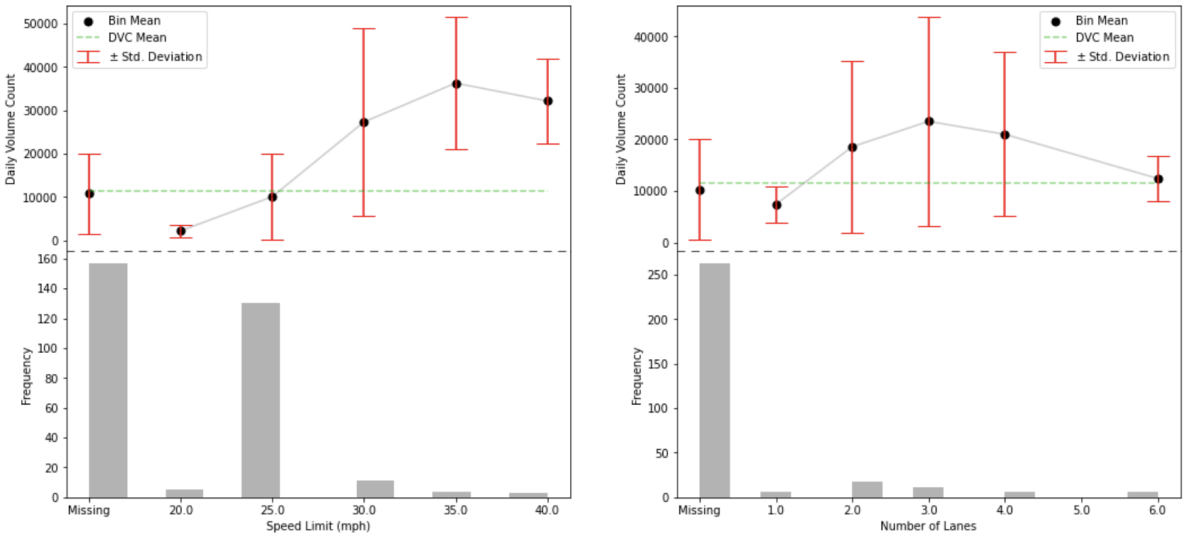


Figure 7: (a) Road length vs. daily volume count, (b) Illustration of normalization.

Here, we introduce a preprocessing method for road length: *normalization*. When the attribute consists of 1-dimensional data, we simply take the mean μ_{train} and standard deviation σ_{train} of the train data, then standardize each example in the dataset (train + test) with the following formula: $x_{norm} = (x - \mu_{train}) / \sigma_{train}$. It is important to find the mean and standard deviation with respect to the train data because we do not want any test data to influence our preprocessing– if it did, then our train data would be a function of our test data, which hinders the legitimacy of any ML algorithm using the dataset. Normalization has several benefits: for example, the data is easily interpretable ($x_{norm} = 0$ is about 50th percentile), and we find that

the centering of data boosts convergence speed in neural networks. Figure 7(b) illustrates how normalization centers the data without altering its underlying distribution.

Speed Limit and ***Number of Lanes:*** the speed limit signals how fast vehicles are legally allowed to go (in miles per hour), and the number of lanes is self-explanatory. These attributes are grouped together because they share similar observations. For example, we hypothesize that each attribute is directly related to traffic volume: vehicles are more likely to take roads with faster speed limits and more lanes. Figure 8 aligns the distribution of each attribute with plots of daily volume count against the SRS attribute values. The distributions indicate that a significant portion of each attribute is missing among the SRS's. Additionally, it appears that the volume count begins to decrease after the speed limit reaches 35mph; likewise, the volume count decreases after the number of lanes reaches 3. These observations suggest that traffic demand may have a limit (around 35mph and 3 lanes) where space on the road begins to outweigh the number of vehicles, leading to smoother traffic flow and consequently less volume.



(a) (b)

Figure 8: (a) Speed limit vs. daily volume count, (b) Number of lanes vs. daily volume count.

The preprocessing methods for speed limit and number of lanes are identical: (1) imputation with the mean of the train data, (2) normalization, and (3) adding a bit indicating whether or not the value was missing. For the first method, if a road segment is missing the attribute value, then the mean (with respect to the train data) is reported in place of the null value. The second method then standardizes the value with the train mean and standard deviation, as discussed before. Lastly, the third method adds a "missing value column" to the dataset, where for each road segment, the column contains 1 if the value was missing, or 0 otherwise. Thus, the two attributes are encoded with 2 values each: one representing the normalized (imputed) value, and the other representing the missing value indicator bit.

3.5. Feature Engineering

The first two engineered attributes–***Direction*** and ***Arctan***– are focused on the orientation of the road segment. Direction has four values– (N,E), (N,W), (S,E), (S,W)– and is determined by the GPS coordinates of the road segment's intersection. Specifically, if (x_1, y_1), (x_2, y_2) are the GPS coordinates of the starting and ending intersections, respectively, then the formula for direction is defined as follows:

$$\text{direction} = (d_{N/S}, d_{E/W}),$$

where

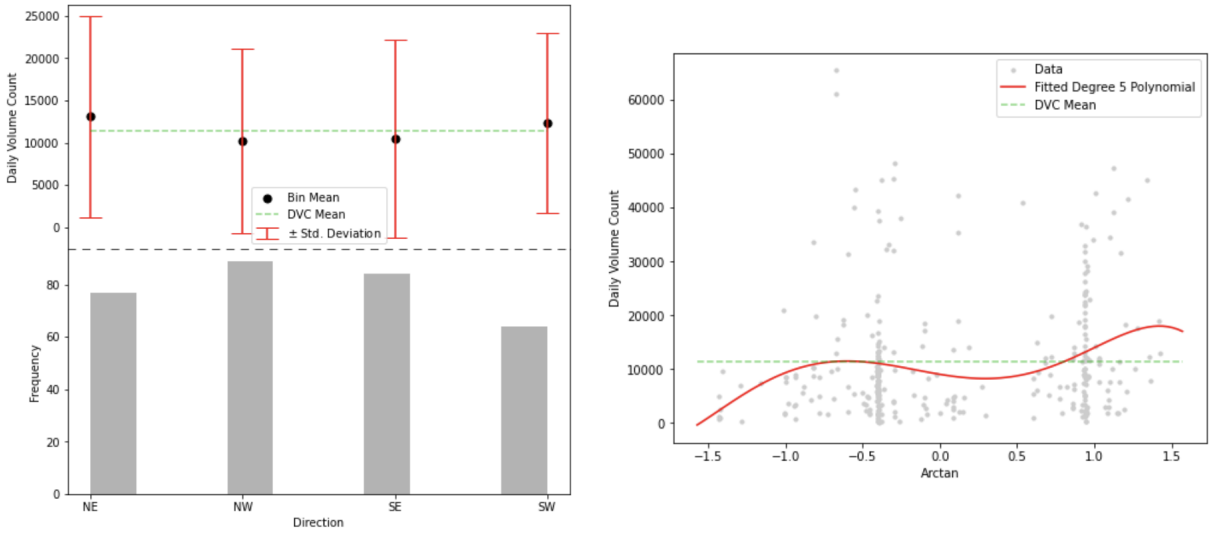
$$d_{N/S} = \begin{cases} N & \text{if } y_2 - y_1 \geq 0 \\ S & \text{otherwise} \end{cases}$$

$$d_{E/W} = \begin{cases} E & \text{if } x_2 - x_1 \geq 0 \\ W & \text{otherwise} \end{cases}.$$

On the other hand, the arctan attribute measures the angle of the road segment, independent of the direction the vehicles move. The equation for arctan is defined below:

$$\text{Arctan} = \begin{cases} \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right) & \text{if } x_1 \neq x_2 \\ \pi/2 & \text{otherwise} \end{cases}.$$

Combining both attributes, we suspect that road segments with similar orientations are more likely to possess similar patterns in traffic volume. Figure 9 presents the distribution of directions, as well as the daily traffic volume plotted against both orientation-based attributes. Immediately, we observe that direction among SRS's is relatively uniform, and that most SRS's have an arctan of about -0.45 or 0.95. This would make sense because Manhattan is a grid, so any pair of road segments are likely to be parallel (same arctan) or perpendicular (arctans differ by $\pi/2$). Moreover, we notice that the four directions have very similar levels of traffic volume, so direction in isolation appears to be a poor predictor of traffic volume; however, the direction



(a) (b)

Figure 9: (a) Direction vs. daily volume count, (b) Arctan vs. daily volume count.

attribute may serve as a useful splitter in decision trees or random forest regressors. Moving to the arctan attribute, the fitted 5th degree polynomial suggests that road segments with a popular arctan value (-0.45 or 0.95) are slightly more likely to have a higher volume count.

Since direction is a categorical variable, no preprocessing is needed, but we must encode the attribute with a vector $v = (v_{NE}, v_{NW}, v_{SE}, v_{SW})$ where each v_i is 1 if the road segment faces direction i, or 0 otherwise. Although arctan is a numerical variable on the continuous interval $(-\pi/2, \pi/2]$, we choose to not preprocess with normalization because the distribution is already zero-centered and the values are already interpretable.

The second pair of engineered attributes– ***Speed Limit ÷ Road Length***, and ***Road Length × Number of Lanes***– are interpretable combinations of existing OSM attributes. The first attribute, speed limit ÷ road length (S÷R), aims to model a metric inversely proportional to travel time, since speed limit (distance / time) is being divided by road length (distance). Specifically the formula is defined as follows:

$$\mathrm{S} \div \mathrm{R} = \begin{cases} \frac{\text{speed\_limit}}{\text{road\_length}} & \text{if speed\_limit exists} \\ \frac{20}{\text{road\_length}} & \text{otherwise} \end{cases}$$

which takes into account missing values in place of the speed limit; this way, the attribute will always be defined. Since vehicles would naturally prefer to take faster roads, we suspect that a metric inversely proportional to time (*i.e.,* S÷R) would be indirectly related to traffic volume. However, observing the relationship between S÷R and daily volume count in Figure 10(a), we notice that S÷R values within a standard deviation of the mean appear to have no distinguishable traffic patterns. With that said, outliers (above a standard deviation away from the mean) tend to reflect lower traffic volume.

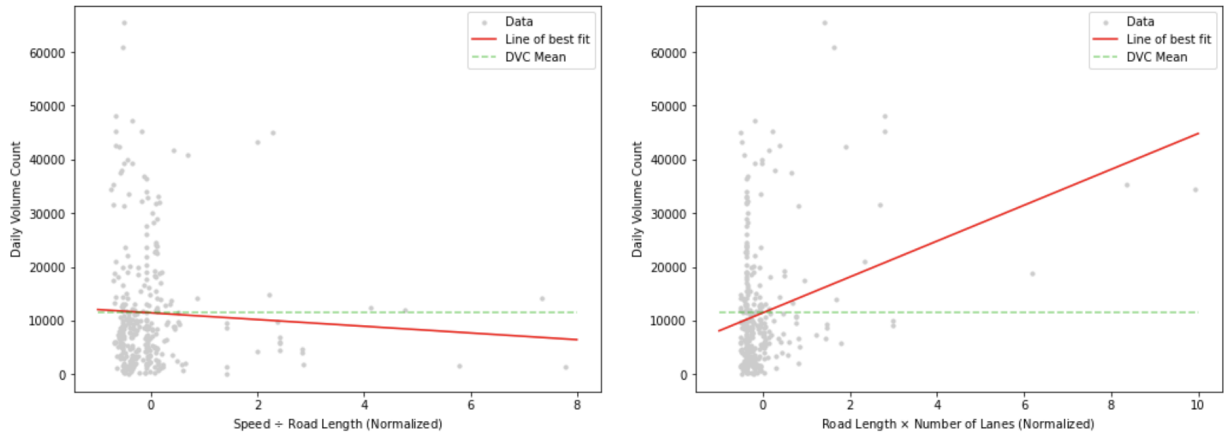


Figure 10: (a) Normalized S÷R vs. daily volume count, (b) Normalized R×L vs. daily volume count.

Similarly, the second attribute, road length × number of lanes (R×L), aims to model a metric representing the amount of space on a road segment. We quantify this measure with the following formula:

$$\mathrm{R} \times \mathrm{L} = \begin{cases} \text{road\_length} \times \text{num\_lanes} & \text{if num\_lanes exists} \\ \text{road\_length} & \text{otherwise} \end{cases},$$

which also takes into account missing values in place of number of lanes. Since long road segments with many lanes (high R×L value) tend to attract more vehicles, we suspect R×L to be directly related to traffic volume. However, the observations from Figure 10(b) are very similar to those of the S÷R attribute; specifically, the outliers above the standard deviation are the only portion of the data serving as a fair predictor of traffic volume. Within a standard deviation of the mean, no pattern appears to exist. As a result, we find both S÷R and R×L (in isolation) to be relatively poor predictors of traffic volume, unless an outlier happens to be sampled. We

preprocess both attributes with normalization, so each is encoded into the feature vector with one representative value– namely, the normalized value from the train mean and standard deviation.

The next two engineered attributes– ***In-Degree*** and ***Out-Degree***– are measures of potential traffic flow through road segments in the network. The definitions are very similar: the *in-degree* of road segment r is the number of road segments (1) adjacent to, and (2) directed at the start intersection of r, while the *out-degree* of r is the number of road segments (1) adjacent to, and (2) directed away from the end intersection of r. Figure 11(a) and 11(b) demonstrate simple examples of in-degree and out-degree, respectively. Furthermore, these attributes are implemented with the **in_degree** ("in_degree — NetworkX 1.10 documentation") and **out_degree** ("out_degree — NetworkX 1.10 documentation") Networkx functions.

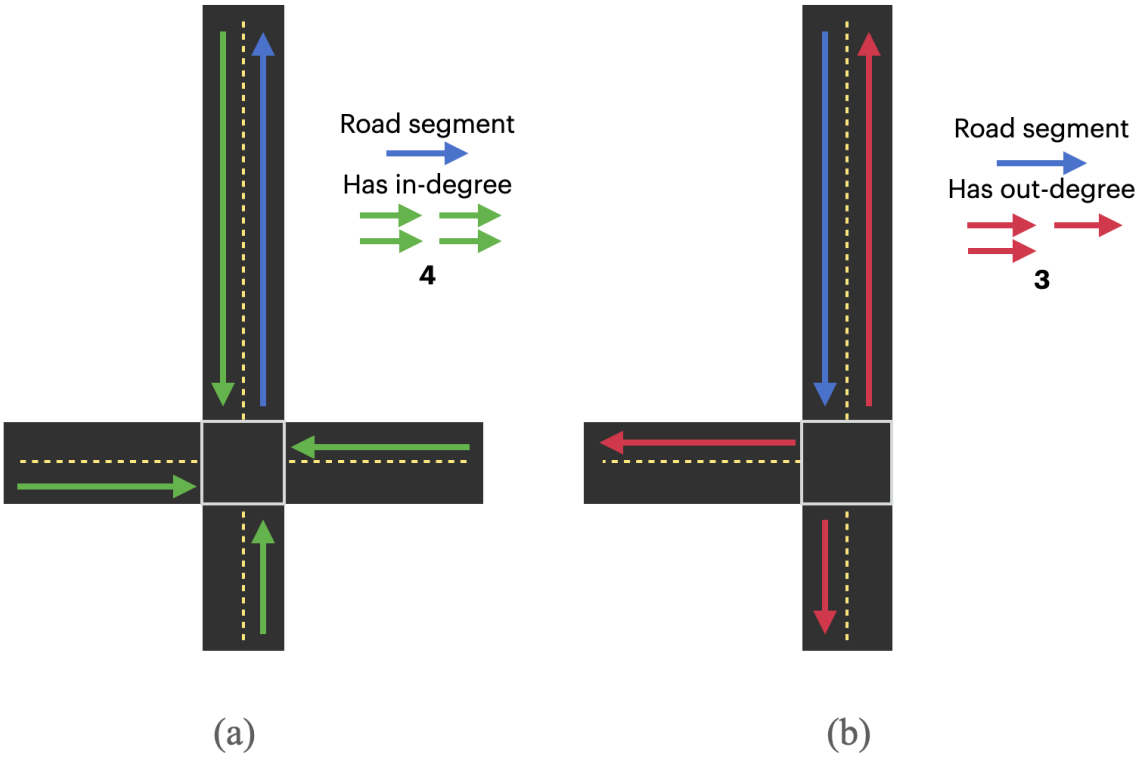


Figure 11: Simple illustrations of (a) in-degree, and (b) out-degree.

We hypothesize that road segments with higher in-degrees have more incoming traffic, thus receive higher levels of traffic volume; similarly, we believe that road segments with higher out-degrees have more reachable destinations, thus receive more traffic and consequently higher levels of traffic volume. Figure 12(a) presents the distribution of in-degrees, as well as the daily traffic volume plotted against the in-degree values; likewise, Figure 12(b) presents the same content for the out-degree attribute. Analyzing the distributions, we notice that both are symmetric and centered at degree 2. Looking at the graph, it appears that an indirect relationship exists between either attribute and daily traffic volume; in both cases, the volume slightly increases past degree 4, but that is likely a consequence of the small amount of degree 5 examples. Overall, our hypotheses look to be the opposite of reality; for both degree types, a higher degree tends to reflect lower levels of traffic volume. Since both attributes are numerical, we choose to preprocess with normalization, therefore encoding each attribute into the feature vector with the normalized values.

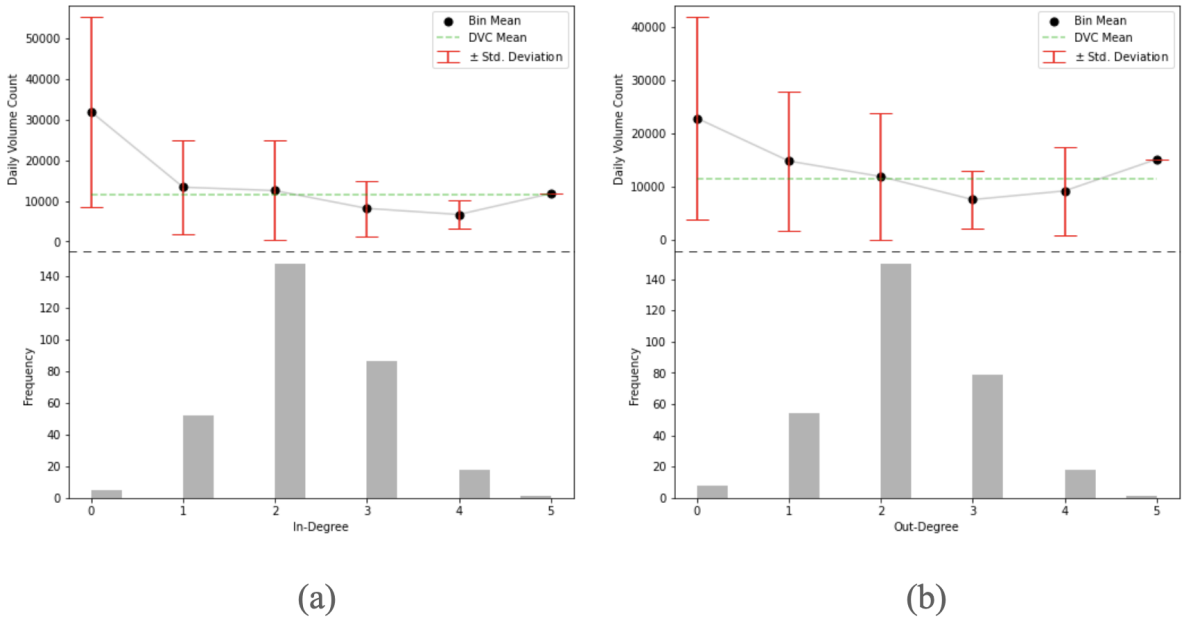


Figure 12: (a) In-degree vs. daily volume count, (b) Out-degree vs. daily volume count.

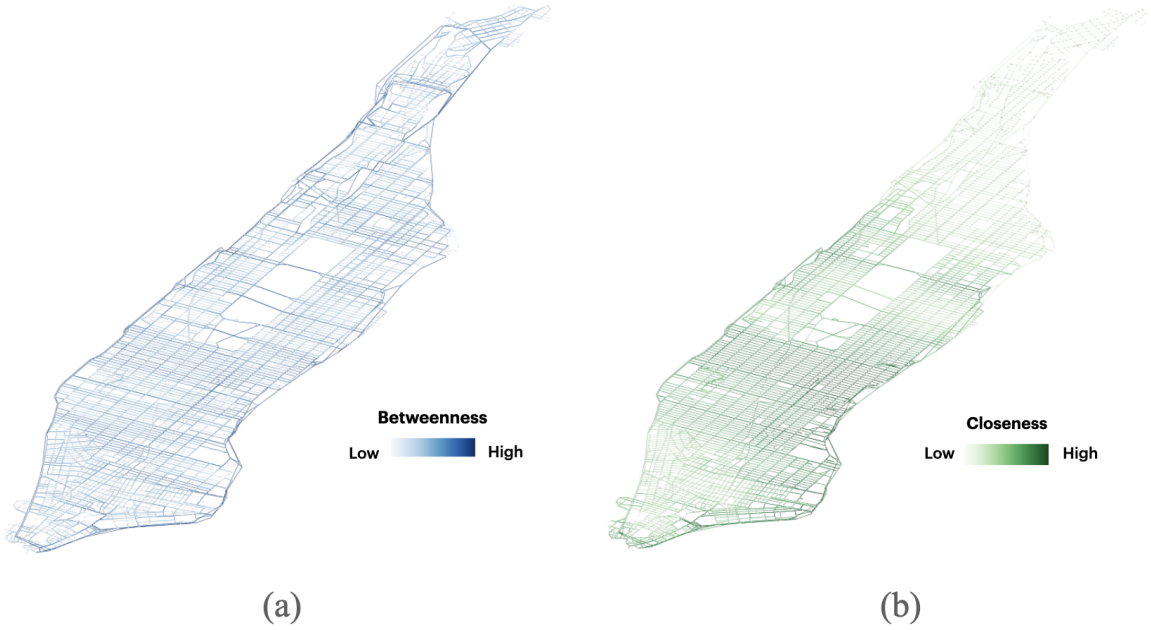


Figure 13: Manhattan road segments color-coded by (a) betweenness centrality, (b) closeness centrality.

The last two engineered attributes– ***Betweenness Centrality*** and ***Closeness Centrality***– aim to measure the importance of road segments (or intersections within the road segment) in the Manhattan network. First, the betweenness centrality (C_B) of road segment r is defined as the percentage of total shortest paths in the network that contain r, which is quantified by the following formula:

$$C_B(r) = \sum_{s \neq t} \frac{\sigma(s,t,r)}{\sigma(s,t)},$$

where $\sigma(s,t)$ is the number of shortest paths between node (intersection) s and node t, and $\sigma(s,t,r)$ is the number of shortest paths between s and t that contain road segment r. Second, the closeness centrality (C_C) of road segment r is the average shortest path length between r and every other road segment in the network, as quantified below:

$$C_C(r) = \frac{1}{|R|-1} \sum_{r' \in R \setminus \{r\}} d(r, r'),$$

where R is the set of all road segments and $d(r, r')$ is the shortest path length between road segments r and r'. Both measures of centrality are illustrated in Figure 13, where darker colors indicate higher values.

We suspect both measures of centrality to behave similarly as predictors of traffic volume; specifically, if a road segment has a high betweenness or closeness centrality, then it must be important to the road network structure, so it should reflect higher levels of traffic volume. Figures 14(a) and 14(b) plot the daily traffic volume against betweenness centrality and closeness centrality, respectively. In both graphs, we notice that centrality values within a standard deviation of the mean appear to follow no clear pattern, but some relatively weak trends are shown by the outliers. However, the betweenness graph reflects a slight *direct* relationship between the outliers and daily traffic volume, while the closeness graph reflects a slight *indirect*

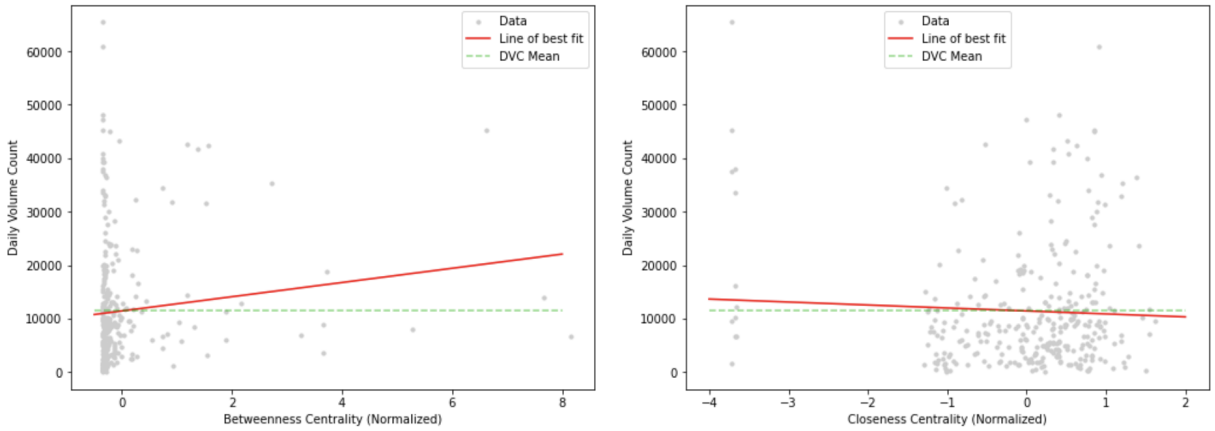


Figure 14: (a) Normalized betweenness vs. daily volume, (b) Normalized closeness vs. daily volume.

relationship. As a result, we conclude that both attributes, in isolation, are poor predictors of traffic volume unless an outlier is sampled. For preprocessing, we use normalization and encode both attributes with one value each– the normalized value.

Overall, we construct our feature vectors with 4 OSM attributes and 8 engineered attributes. Because some attributes are encoded with more than one value (*e.g.,* GPS coordinates, speed limit, direction), each feature vector will possess a length longer than the total number of attributes (12). Specifically, we encode the feature vectors with 36 values, which is illustrated in Figure 15 below. The feature matrix will therefore have a size of 310×36, while the target matrix will be 310×24.

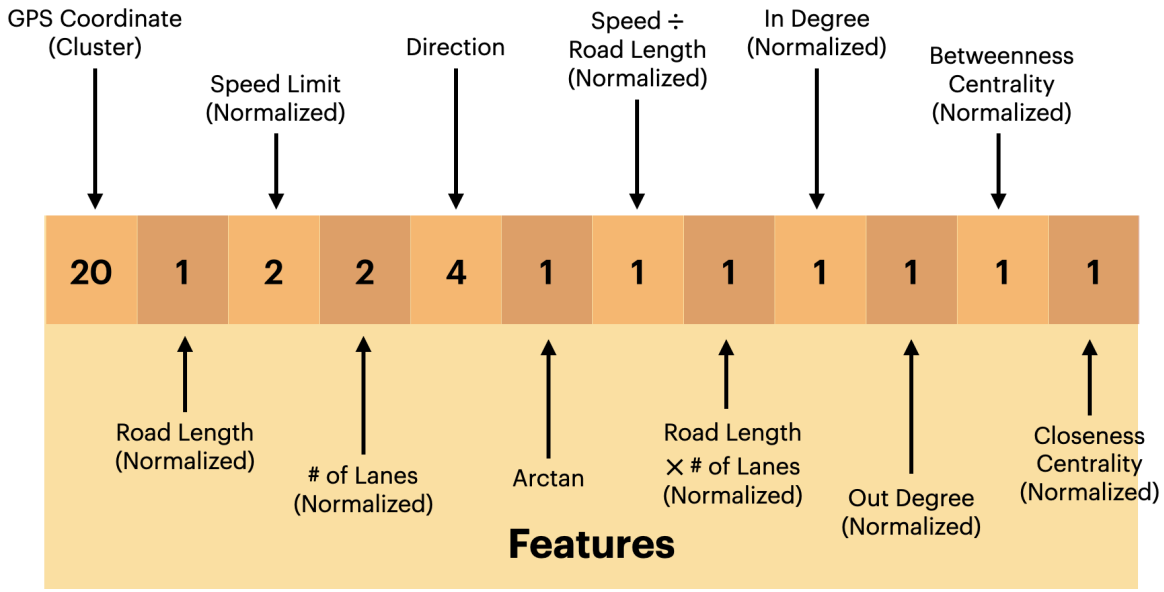


Figure 15: Feature vector encoding breakdown by attribute.

3.6. Cross-Validation

Having discussed the generation of feature-target pairs in our dataset, the next step is preparing the examples for predictive modeling. Typically, when strictly concerned with the

development of a ML model, we partition the feature-target examples into two sets: one for training the model (*e.g.,* 90% of the examples), and one for testing the model (*e.g.,* the other 10%). This way, we can evaluate whether or not the model generalizes to unforeseen examples– an important property for any effective, deployed model. However, this method (reflecting a single train-test split) introduces problems when noisy examples populate the test set; specifically, when the dataset is relatively small (*e.g.,* 310), the test set is likewise very small

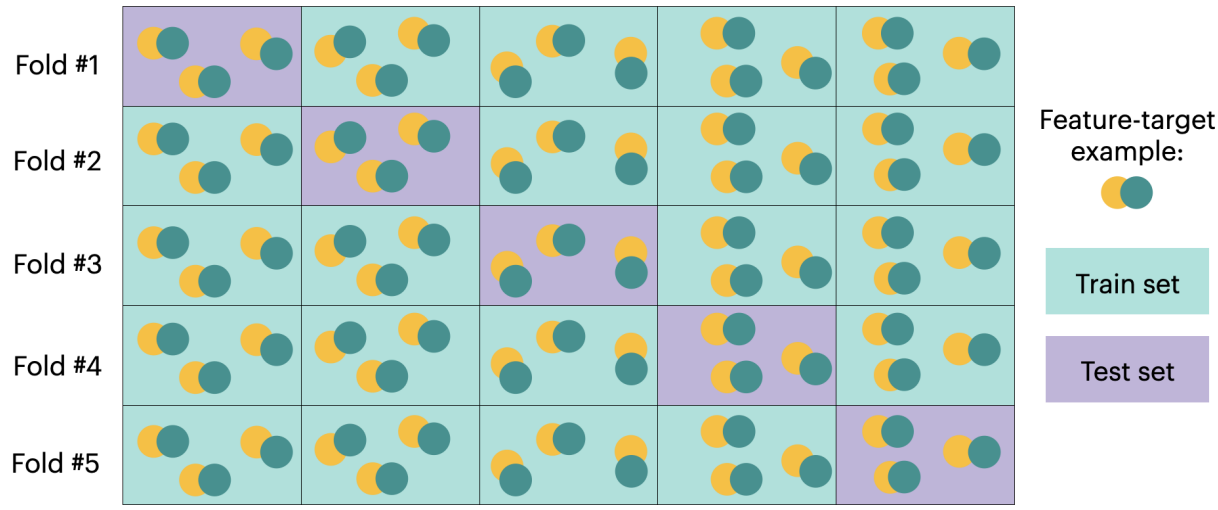


Figure 16: Visual example of 5-fold cross validation.

(*e.g.,* 31) so the possible assignment of noisy examples to the test set will hinder its ability to represent the underlying data-generating distribution encountered in deployment. Addressing this problem, robust methods have been developed to properly evaluate the test performances of ML algorithms: one popular example is *K-fold cross validation*, which is illustrated in Figure 16.

Instead of evaluating on a single train-test split, the *K*-fold cross validation technique considers *K* different train-test splits (*a.k.a.*, folds). Specifically, the *K* folds are curated so that the *K* test sets are mutually exclusive and span the whole dataset. These two properties ensure

that every train-test split consists of $\sim 100(1 - \frac{1}{K})\%$ train examples and $\sim 100(\frac{1}{K})\%$ test examples. Lastly, the *K*-fold cross validation technique evaluates the performance of a given model by (1) training and testing it on each fold, then (2) reporting the mean test error– and sometimes the standard deviation too– over the *K* folds. In our experiments, we use 10-fold cross validation (with a 90:10 train-test split) over the default 5-fold cross validation (with a 80:20 split) because our dataset is small, so we need to reserve most of the examples for training.

4. PREDICTIVE MODELING

In this section, we detail the implementation and hyperparameter tuning processes of each ML algorithm– K-Nearest Neighbors, Decision Tree, Random Forest, and Neural Network– considered for solving the task of traffic volume prediction. First, we provide an overview of the ML pipeline, followed by a discussion on the mean absolute error (MAE), an interpretable and robust performance metric. Then we cover the specifics of each ML algorithm and analyze the hyperparameter tuning results. Lastly, we compare the best-performing instances of each ML algorithm with respect to test error, train time, and test time.

4.1. Modeling Overview

We begin with an overview of the ML pipeline, followed by a brief discussion on the ML algorithms we choose for the task at hand. In general, we consider an ML algorithm to be a function defined by several configurations (*e.g.,* model, optimization technique, regularization) that receives a dataset and outputs a "predictor" function intended to best fit the dataset. Although much of a ML algorithm is typically fixed, there typically exist *hyperparameters*– values that help define the model architecture, optimizer, *etc.,* to learn on the dataset. We bring

light to hyperparameters because they help ML algorithms adapt to a greater diversity of datasets– one dataset may perform better with one set of hyperparameter instances, while another dataset may perform better with a completely different set. Since our dataset is fixed, we are primarily concerned with the hyperparameter tuning of each algorithm we use; specifically, for each ML algorithm, we define a set of possible hyperparameter settings H (elements in the set can be multidimensional) and evaluate the performance of the algorithm tuned with each setting $h \in H$ via 10-fold cross validation. Then, we select the hyperparameter setting $h^* \in H$ that yields the best performance. Finally, the ML algorithm defined by h^* outputs a predictor function that predicts the 24-hour traffic volume vectors for *all road segments* in Manhattan. Figure 17 illustrates the ML pipeline just described; in the following subsections, we will clarify the hyperparameter specifics for each ML algorithm, followed by a reporting of the hyperparameter spaces and the best-performing values.

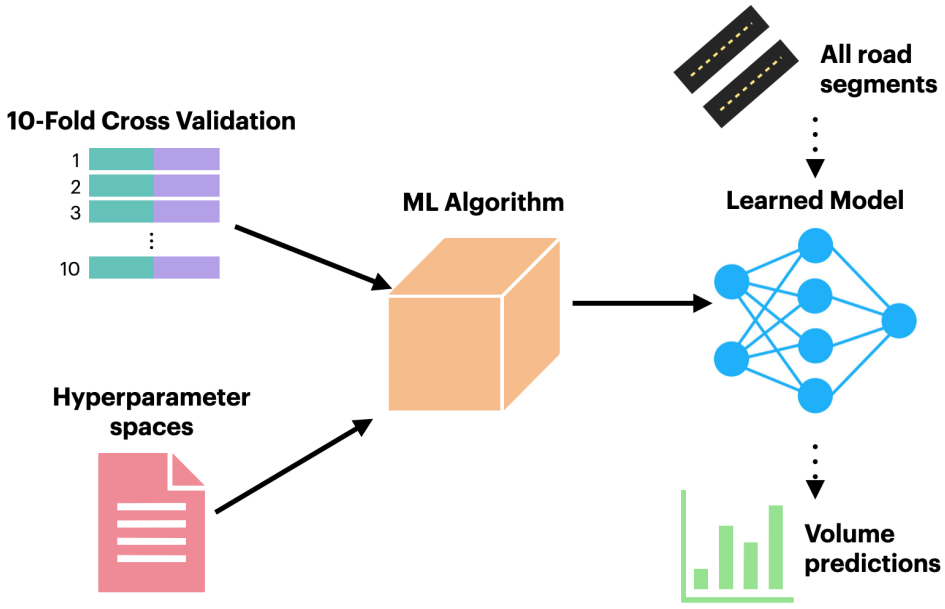


Figure 17: Illustration of the ML pipeline used in the project.

So far, we have discussed ML algorithms in general terms: next, we move to specific examples of ML algorithms, paired with the hyperparameters in consideration. The tree in Figure 18 represents the hierarchy of ML algorithms used in our experiments. We begin with the root of the tree– supervised learning algorithms, or algorithms that learn on datasets composed of input-output pairs. Moving down the tree, we partition the supervised learning algorithms into two types: instance-based and model-based learning algorithms. Instance-based algorithms construct a predictor function that *directly uses* the input-output pairs in the train set, while model-based algorithms construct a predictor function defined by parameters that *implicitly embed* the training pairs (Evans). The two types differ in both time and memory: instance-based algorithms typically require no training time but more testing time and memory, while model-based algorithms require more training time but less testing time and memory (Evans).

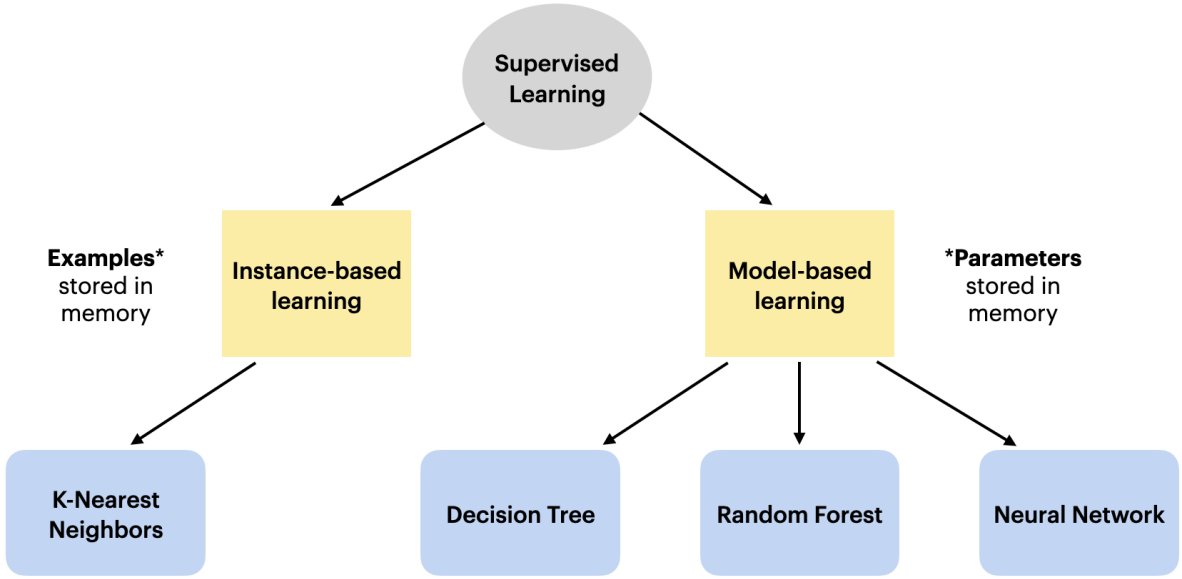


Figure 18: Hierarchy of supervised learning algorithms.

For the sake of completeness, we consider both types of ML algorithms; specifically, *K-Nearest Neighbors* for instance-based learning, and *Decision Tree*, *Random Forest*, and *Neural Network* for model-based learning. Each algorithm is described in its own subsection, which includes the implementation details, hyperparameter tuning procedure, reporting of quantitative results, and visualization of qualitative results.

4.2. Performance Metric

Before detailing the implementation of each ML algorithm, it is important to establish a performance metric that best quantifies the overall similarity between each pair of ground-truth volume vector V and predicted volume vector $\widehat{V}$. In the following experiments, we primarily consider the *mean absolute error* (MAE) defined by the following formula:

$$MAE = \frac{1}{24n}\sum_{i=1}^{n}\sum_{j=1}^{24}\left|v_j^{(i)} - \hat{v}_j^{(i)}\right|,$$

where $v_j^{(i)}$ and $\hat{v}_j^{(i)}$ are the traffic volume counts for hour j of the i'th pair of ground-truth and predicted volume vectors in the test set. In the task of multi-output regression (estimating real-number vectors), another common performance metric is the *mean squared error* (MSE), where the square $(\bullet)^2$ operator is substituted in place of the absolute value operator $|\bullet|$ in the above equation. We justify the choice of MAE over MSE with a couple reasons. First, the MAE is more interpretable in the context of our problem: simply, the MAE is the average hourly difference in actual vs. predicted volume counts. For example, a MAE of 100 means that the model is (on average) off by 100 vehicles per hour. Second, the MAE metric is more robust to outliers compared to the MSE: it is possible that our dataset contains noisy examples, so we prefer to relax the "punishment" inflicted upon the model for wrongly predicting noise in traffic

volume. On the other hand, the MSE is more sensitive to outliers, which tends to encourage models to focus on noisy examples rather than the practical ones. Since the MAE metric is a measurement of *error*, a well-performing model will yield a smaller MAE, and vice-versa.

Interpreting the scale of MAE is difficult without proper context of the task at hand; depending on the problem, a model yielding a MAE of 200 may be more impressive than one yielding a MAE of 2. Generally, it is sufficient to compare performance metrics given by ML predictors to the *baseline performance metric*– the theoretically-optimal performance of a predictor that strictly has access to the *targets*, not features. This metric is relevant under the assumption that no pattern exists between the features and targets, so if a model yields a performance metric *better* than the baseline, then we have reason to believe that features predict targets to at least some extent. Therefore, when we consider the performance of ML algorithms, we compare these results to the baseline. Specifically, if $v^{(1)}, v^{(2)}, \dots, v^{(n)}$ are the volume vectors in the test set, then we define the *baseline MAE* as follows:

$$BMAE = \frac{1}{24n}\sum_{i=1}^{n}\sum_{j=1}^{24}\left|\bar{v}_j - v_j^{(i)}\right|,$$

where $\bar{v}$ is the *element-wise median* of the volume vectors in the train set. We do this because the following property holds (Schwertman, et al.):

$$\bar{v} = \text{median}(\mathcal{V}_{\text{train}}) = \text{argmin}_{v' \in \mathbb{R}^{24}} \text{MAE}\left(\{(v^{(i)}, v')\}_{i \in [n]}\right).$$

A similar relationship exists between the MSE metric and the element-wise mean of the train set vectors. To demonstrate the difference between the element-wise median and mean predictors with respect to MAE, Table 2 reports the specific baseline errors of the traffic volume dataset. In the following subsections, the smaller (median) baseline error will be compared to the errors given by each ML model.

Baseline Predictor	MAE (mean ± std)
Mean	353.9 ± 56.5
Median	325.9 ± 57.1

Table 2: A comparison of the baseline predictors with respect to the MAE.

4.3. K-Nearest Neighbors

Our first ML algorithm is the sole example of instance-based learning in consideration. As the type implies, no training is required; instead, the set of train examples $\{(x_i, y_i)\}_{i \in [n]}$ is stored in memory. For testing, the K-Nearest Neighbors (KNN) algorithm is executed as follows: first, a feature input x_{new} is received with the goal of predicting y_{new}. To do so, we calculate $d(x_{new}, x_i)$ for each x_i in the train set with some distance metric d. Then we find the K examples– $(x_1^*, y_1^*), (x_2^*, y_2^*), \ldots, (x_K^*, y_K^*)$ – yielding the smallest distances. Lastly, we predict y_{new} with $\widehat{y}_{new}$, the element-wise median across $y_1^*, y_2^*, \ldots, y_k^*$. Figure 19(a) illustrates a simple example of the KNN algorithm.

The two parameters defining the KNN algorithm are the distance metric d and number of neighbors K. In many cases, d is set to the Euclidean distance (*a.k.a.,* L2 norm) defined below:

$$d\left(x^{(1)}, x^{(2)}\right) = \left\|x^{(1)} - x^{(2)}\right\|_2 = \sqrt{\sum_{i=1}^{24} (x_i^{(1)} - x_i^{(2)})^2}.$$

However, the context of our task deals with a *road network* as opposed to Euclidean space. Therefore, we stray from the conventional choice of d in favor of a new metric: length of shortest path between two edges in the road network. Specifically, the features in the KNN

algorithm are *directed edges* (road segments) in the Manhattan network, and the adjusted distance metric $d(e_1, e_2)$ is the length of the shortest path between edges e_1 and e_2. Figure 19(b) demonstrates a simple example of the calculated distances between a road segment and three others. Since we are concerned with shortest path lengths to compute distances, we implement the KNN algorithm using Networkx's **single_source_djikstra** function ("single_source_dijkstra — NetworkX 1.10 documentation").

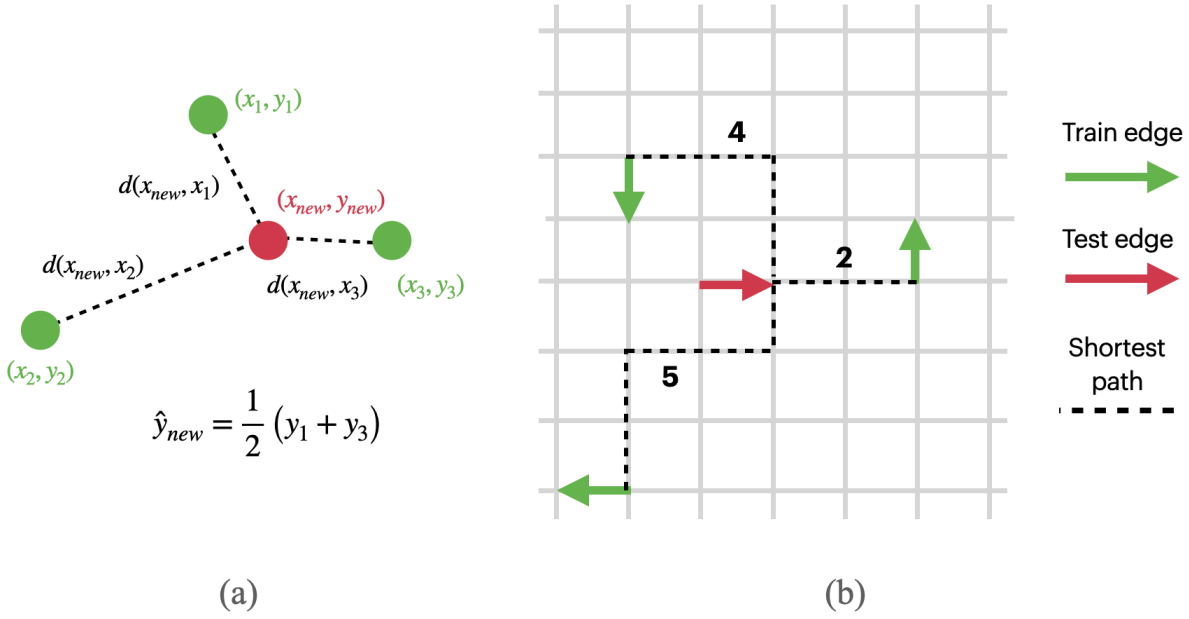


Figure 19: (a) Illustration of the KNN algorithm, (b) Simple computations of modified distance metric.

On the other hand, the number of neighbors K will be our sole hyperparameter for the algorithm. Specifically, we consider the values $K \in$ 1 to 20 in steps of 1, then 25 to 100 in steps of 5. Figure 20 highlights the performance of each K value in the search space; the train and test MAEs are reported in the form *mean* ± *standard deviation* over 10-fold cross validation. These numbers are also reported in Appendix A.1. The best-performing result $(K = 10)$ yields

an error of 309. 0, which lies below the baseline MAE of 325.9. The values of K below 10 appear to produce models that *overfit*– a phenomenon where the model fails to generalize its predictive capability to the test set because it focuses too narrowly on individual train set examples. For example, a KNN instance of $K = 1$ will solely consider the closest data points in the train set; this clearly yields a train error of 0, but does not effectively generalize to the test set

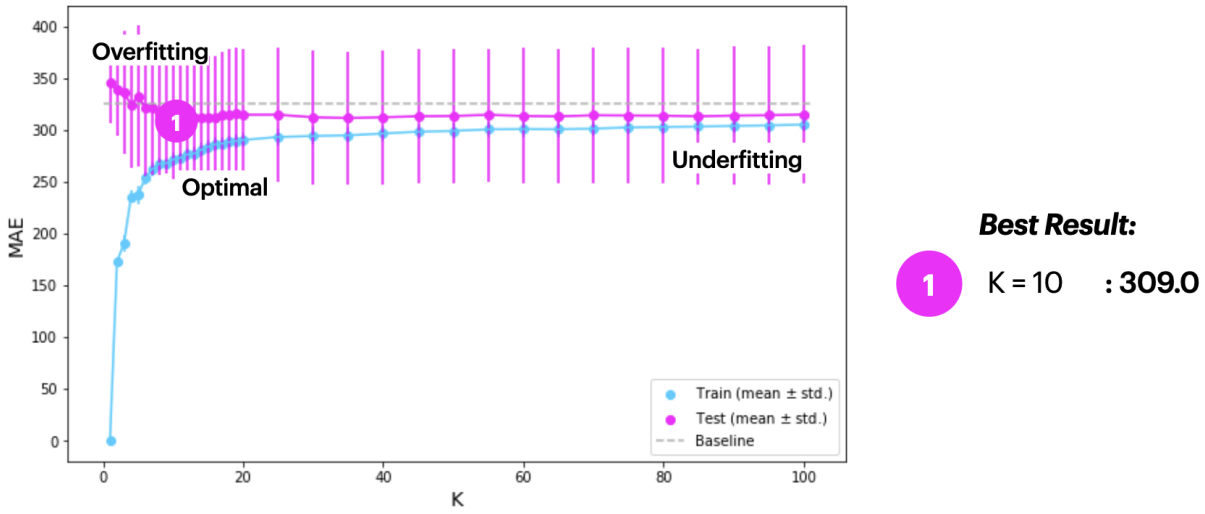


Figure 20: MAE results of hyperparameter tuning the KNN algorithm.

because the model gives unreasonable weight to noisy points. Conversely, the values of K well above 10 appear to produce models that *underfit*– a phenomenon where the model is provided very limited information about the individual train examples, making it hard to differentiate between test examples. In this case, high values of K will approach the baseline prediction method– taking the element-wise median of the train data; this is reflected in Figure 20, where the train and test errors of high K values approach the baseline MAE.

As stated before, we are also interested in the way that each ML model gives predictions for all road segments in the Manhattan network. In doing so, we take snapshots of the traffic

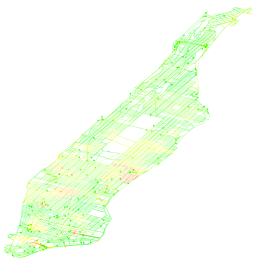
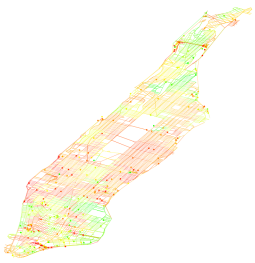
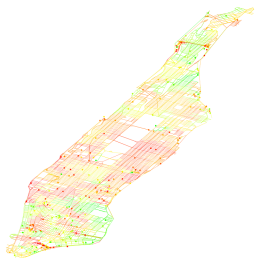
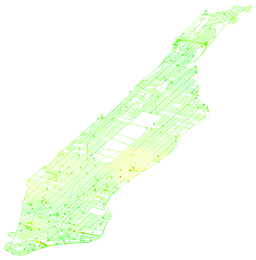
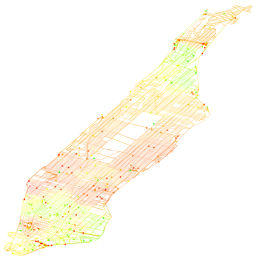
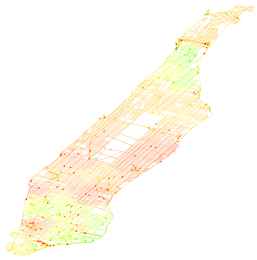
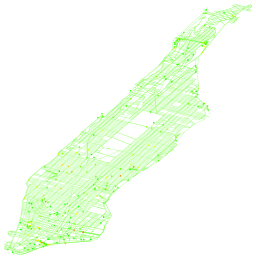
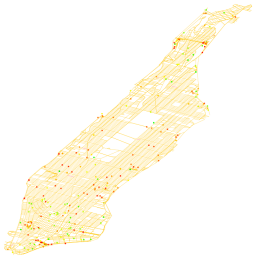
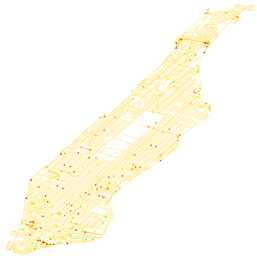
Case	K	3:00AM	11:00AM	7:00PM
Overfit	1			
Optimal	10			
Underfit	150			

Table 3: Visualization of KNN traffic volume predictions.

volume predictions of three KNN models at the times 3:00AM, 11:00AM, and 7:00PM, which are provided in Table 3. The three models are defined by $K = 1$ (overfitting), $K = 10$ (optimal), and $K = 150$ (underfitting). In the overfitting case, the predictions appear to be more aggressive: many road segments are very red and many are also very green. This is typically a consequence of placing too much weight on single noisy examples. In the underfitting case, the

predictions are much more uniform: the vast majority of road segments are predicted to have the same traffic volume counts. A balance between the two extremes is shown by the $K = 10$ case, where none of the predictions appear out-of-ordinary, but clusters are still formed.

4.4. Decision Tree

Our next ML algorithm is the first example of model-based learning. Specifically, a decision tree is composed of nodes and leafs (*i.e.,* the end nodes): nodes represent *decisions* that split the relevant data into two or more subsets, while leafs represent the final *predictions*. Since the task is multi-output regression, leafs will take the form of 24-length vectors. The root begins with the whole train set, and as we move down the tree, the train set is split into more and more groups of data; once a leaf is reached, the remaining group is assigned the prediction corresponding to the leaf. Figure 21(a) provides a simple example of a decision tree. The question that remains is: how do we build a *well-performing* decision tree from the train set examples? Generally, the algorithm begins at the root with a best attribute split– a split in the dataset that will group the train examples by similar patterns in traffic volume. Two methods are used for this: the "Best" method looks for a single attribute split that maximizes information gain (with respect to MAE), while the "Random" method randomly selects an attribute split from the ones that achieve any information gain. Although the "Best" method seems ideal, it can lead to overfitting because of its greedy nature; therefore, in many cases the "Random" method is slightly prefered. Once the root data is split into groups, the algorithm is recursively applied to each group. This ultimately leads to the partition of the train data into many groups that share similar attributes and patterns in traffic volume. Once the final groups are formed, each leaf is

assigned the traffic volume average (with respect to the mean or median, depending on the criterion) of one group.

We implement the decision tree algorithm with Scikit-Learn's **DecisionTreeRegressor** class ("sklearn.tree.DecisionTreeRegressor — scikit-learn 1.1.1 documentation"), which handles multi-output regression. Specifically, the criterion we use is "MAE" and the splitting method is "Random". Building a decision tree to perfectly fit a dataset is possible, but an ill-posed idea; if the final groups are too small, then the model is destined to fail on the test data. This is due to the

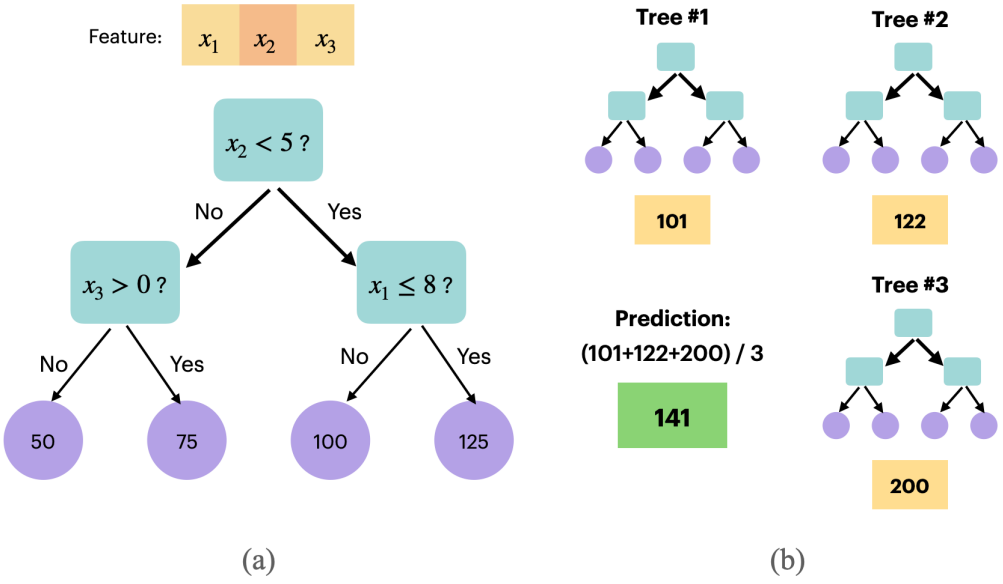


Figure 21: Simple examples of (a) decision tree, and (b) random forest algorithms.

fact that small groups (even groups of one example) contain data reflecting noisy counts in traffic volume. To mitigate the possibility of overfitting, we consider a pair of regularization methods via *pre-pruning* (carefully limiting the construction of the tree). The first method is placing a limit on the depth of the tree: this ensures that the final groups are larger, thus less prone to overfitting influenced by noisy data. We do this by tuning down the **max_depth** hyperparameter

in Scikit-Learn's implementation; however, if this value is too small, the decision tree will have smaller capacity and will likely underfit on the train data. The second regularization method is restricting the amount of attributes considered for splitting: this encourages a less-greedy building technique by placing more weight on attribute splits with smaller information gain. We do this by tuning down the **max_features** hyperparameter– the percentage of total attributes considered for splitting at each node– in Scikit-Learn; however, if this value becomes too small, then the algorithm is restricted in its optimization ability, which leads to underfitting.

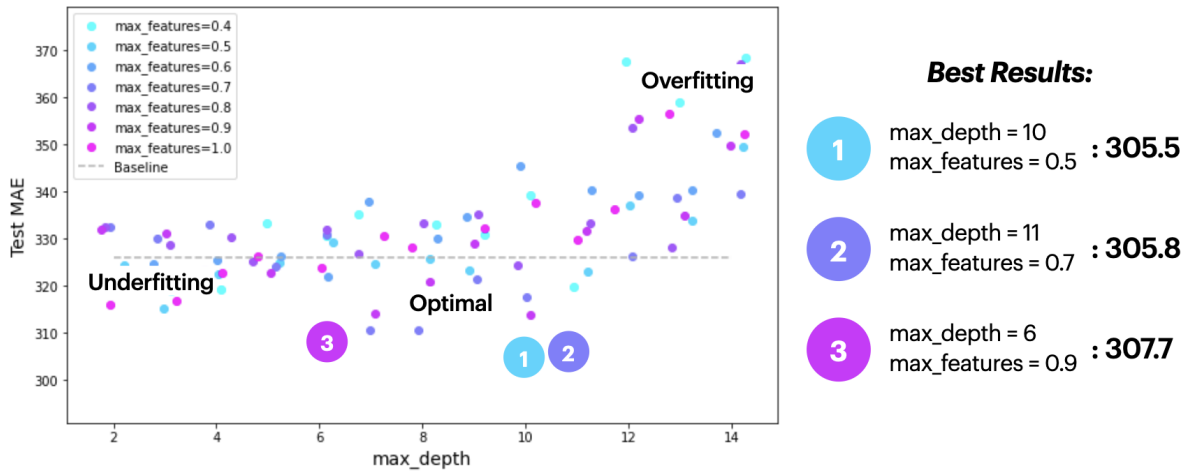


Figure 22: MAE results of hyperparameter tuning the decision tree algorithm.

In our decision tree experiments, we hyperparameter tune on the values **max_depth** $\in$ 2 to 14 in steps of 1, and **max_features** $\in 0.4$ to 1.0 in steps of 0.1. Figure 22 highlights the test performance of each hyperparameter setting in the search space. These numbers are also reported in Appendix A.2. Looking at the graph, it seems to reflect that no one particular **max_features** value gives the best results, but the range 6 to 11 appears to work the best for **max_depth**; the top three results fall into this range, but have varying **max_features** values (0.5, 0.7, 0.9). Our

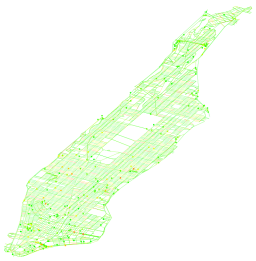
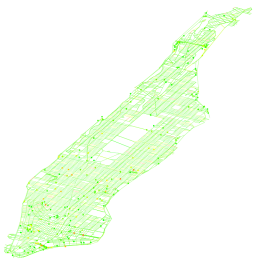
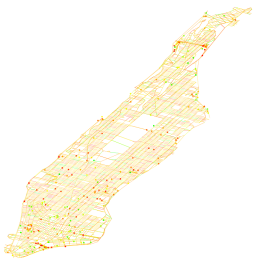
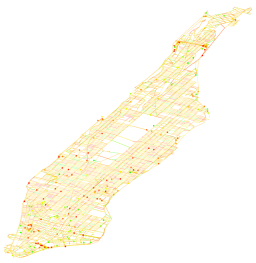
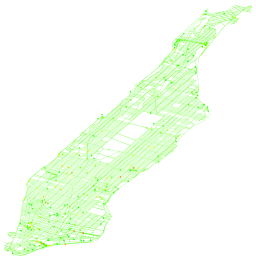
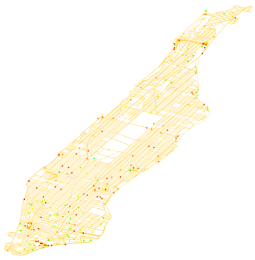
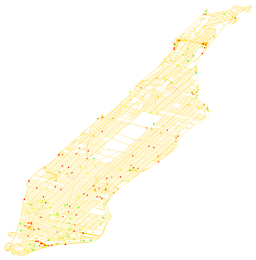
Case	H	3:00AM	11:00AM	7:00PM
Overfit	$\frac{20}{1}$			
Optimal	$\frac{10}{0.5}$			
Underfit	$\frac{1}{1}$			

Table 4: Visualization of decision tree traffic volume predictions. Settings in the hyperparameter space H take the form: **max_depth** | **max_features**.

best-performing result yields a test MAE of 305.5 , which is 20.4 less than the baseline error and 3.5 less than the best KNN error. For **max_depth** values over 11, the decision tree models appear to *overfit*, since the final groups are too small, thus strongly influenced by noisy

examples. Conversely, the **max_depth** values under 6 produce decision tree models that appear to *underfit*, since the final groups are too large and embed little information about the relationship between attributes and traffic volume.

In addition, we provide snapshots of the traffic volume predictions from three decision tree models at 3:00AM, 11:00AM, and 7:00PM in Table 4. Specifically, we include an overfitting case (high **max_depth** and low **max_features**), optimal case (mid **max_depth** and **max_features**), and underfitting case (low **max_depth** and **max_features**).
The same observations can be made for decision tree predictions as for KNN ones: the overfitting case reflects aggressively-colored predictions, the underfitting case reflects uniform, same-colored predictions, and the optimal case is an effective balance between the two extremes. One notable difference is that the predictions given by the optimal decision tree model are visually much different than those given by the optimal KNN model; the latter tends to reflect geographic clusters, while the former seems almost random. This is a consequence of the attributes used: the KNN model is strictly concerned about the location of road segments, while the decision tree model incorporates a variety of non-location attributes.

4.5. Random Forest

Our next ML algorithm– *random forest regression*– is an extension of the decision tree algorithm. Having established an understanding of decision trees, the promotion to random forest regression is relatively straightforward. In the training stage, a "forest" of decision trees is created; specifically, each tree is built using "Random" splitting, so the node and leaf content will vary across the trees. As mentioned in the previous subsection, the decision trees are individually optimized to accurately predict the traffic volume in the train data. For testing, a

feature is fed into each decision tree, which returns a set of predicted volume vectors. The final predicted volume vector for the feature is the element-wise *mean* or *median* (we use median since we deal with MAE) of each tree's individual prediction. Figure 21(b) illustrates a simple example of a random forest, where the final prediction is the mean of the 3 trees' predictions. We implement the random forest algorithm with Scikit-Learn's **RandomForestRegressor** class ("sklearn.ensemble.RandomForestRegressor — scikit-learn 1.1.1 documentation"), which handles multi-output regression; additionally, the criterion is set to MAE.

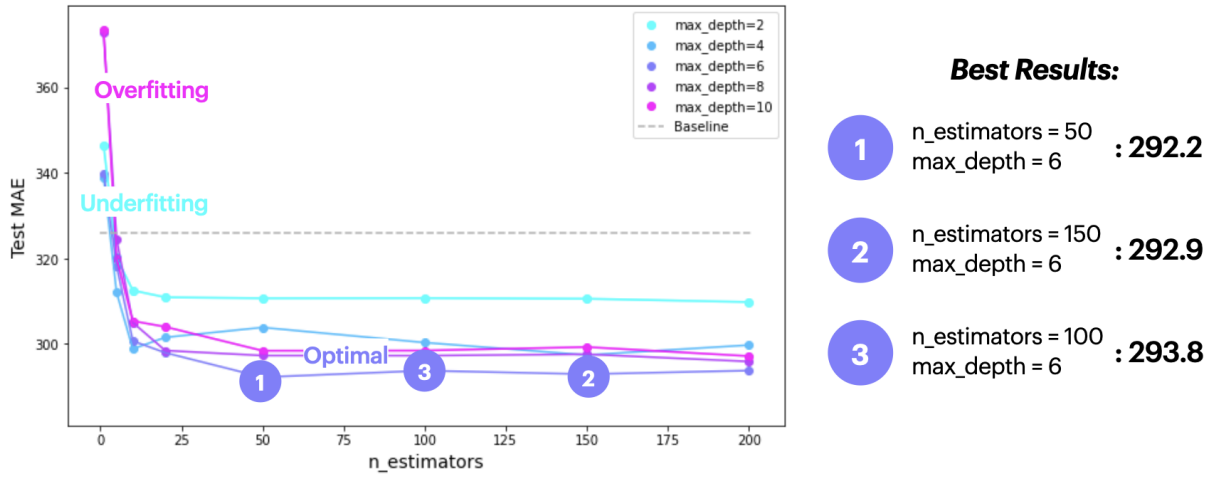


Figure 23: MAE results of hyperparameter tuning the random forest algorithm.

Like decision trees, random forests are susceptible to overfitting on the train data; to mitigate this possibility, we consider two regularization methods. The first method is simply to add more trees (*a.k.a.,* predictors, or estimators) to the forest. Since the output of the random forest algorithm is determined by querying an ensemble of individually-intelligent predictors, it follows that increasing the number of predictors will yield more accurate results; specifically, we do this by tuning up the **n_estimators** hyperparameter in Scikit-Learn's implementation. The

second method is to place a limit on the depth of each tree; this is done by tuning down the **max_depth** hyperparameter. As in the decision tree algorithm, this hyperparameter forces each tree in the random forest algorithm to partition the train set into larger groups, which in turn are less susceptible to noisy examples.

In our random forest experiments, we hyperparameter tune on the values **n_estimators** $\in \{2, 5, 10, 20, 50, 100, 150, 200\}$ and **max_depth** $\in$ 2 to 10 in steps of 2. Figure 23 highlights the test performance of each hyperparameter setting in the search space; these numbers are also reported in Appendix A.3. Analyzing the graph, the tuning of **max_depth** $= 6$ clearly performs the best, while every **n_estimators** value above 20 appears to perform very similarly. These observations are reflected by the top three results, which all have a **max_depth** of 6 and varying **n_estimators** values above 20. Specifically, the top result yields a test MAE of 292.2, which is 33.7 less than the baseline error and 13.3 less than the best decision tree error. For **n_estimators** < 5 combined with **max_depth** > 6, the random forest models appear to *overfit*, since the final groups in each tree are too small, thus strongly influenced by noisy examples. On the other hand, **n_estimators** < 5 combined with **max_depth** < 6 produce random forests that appear to *underfit*, since the final groups in each tree are too large to embed sufficient information about the relationship between the attributes and traffic volume counts.

Similar to the previous two algorithms, we include snapshots of the traffic volume predictions from three random forest models at 3:00AM, 11:00AM, and 7:00PM in Table 5. The overfitting case reflects low **n_estimators** and high **max_depth**; the optimal case reflects high **n_estimators** and mid **max_depth**; and lastly, the underfitting case reflects low **n_estimators** and low **max_depth**. The same observations can be made for random forest predictions as for the preceding ones: the overfitting case reflects aggressively-colored predictions, the underfitting

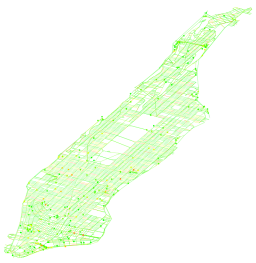
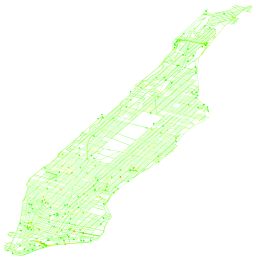
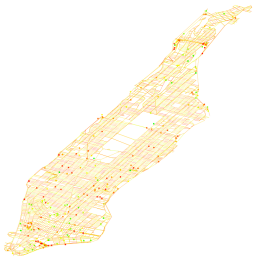
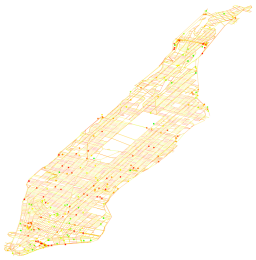
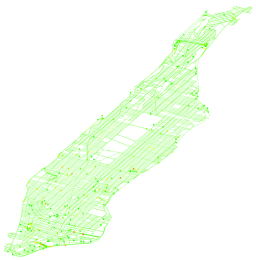
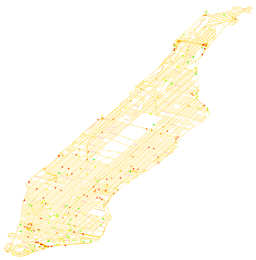
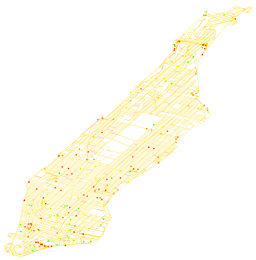
Case	H	3:00AM	11:00AM	7:00PM
Overfit	$\frac{1}{20}$			
Optimal	$\frac{50}{6}$			
Underfit	$\frac{2}{1}$			

Table 5: Visualization of random forest traffic volume predictions. Settings in the hyperparameter space H take the form: **n_estimators** | **max_depth**.

case reflects uniform, same-colored predictions, and the optimal case is an approximate balance between the two extremes. Another noteworthy consideration of the random forest algorithm is the extent (or degree) each attribute contributes to the prediction of hourly traffic volume. In doing so, we appeal to Scikit-Learn's implementation of *feature importance* ("sklearn.ensemble.

RandomForestRegressor — scikit-learn 1.1.1 documentation"), the quantification of "importance" across the encoded attributes in the feature vector. Specifically, if a value belonging to an encoded attribute is given a high (low) measure of feature importance, then the algorithm relies heavily (moderately) on the encoded attribute for making accurate traffic volume predictions. Figure 24 highlights the importance of each value in the feature vector, organized by the attribute groups presented in Section 3. Overall, the bar graph reassures that most attribute groups contain information helpful to the optimization of random forests; for example, the top 5 important values come from 4 different attribute groups. The one attribute group that yields relatively low importance– In-Degree + Out-Degree– may seem unnecessary to keep in the feature vector, but the inclusion of these attributes helps improve the performance of neural networks, as discussed in the next subsection. These results also suggest that linear correlation (or the lack thereof) between attributes and daily traffic volume are not always indicative of how useful the attribute will be in a non-linear model, such as random forest regression.

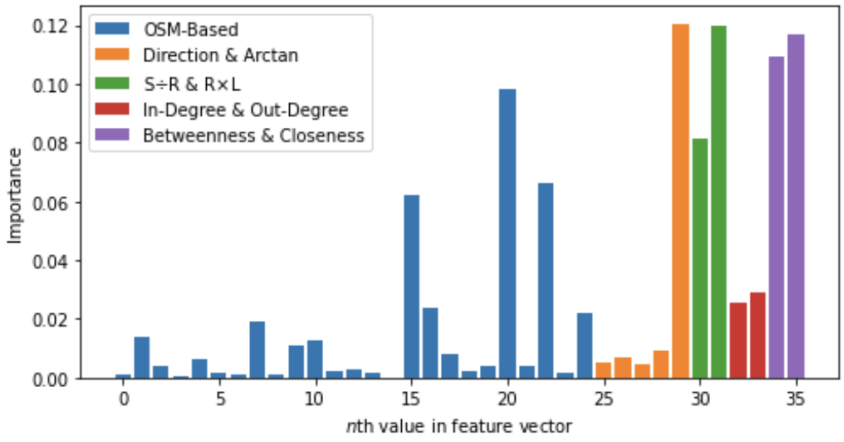


Figure 24: The importance of values across the feature vector in the random forest algorithm.

4.6. Neural Network

Our last ML algorithm– neural network– is also an example of model-based learning. Specifically, the model is composed of linear layers organized in a chain and separated by activation functions (*e.g.,* sigmoid, ReLU, tanh); the first layer receives the input feature and the final layer represents the output, *a.k.a.* the 24-hour volume vector. Figure 25(a) gives an example of a neural network architecture used in our experiments. Each linear layer is parameterized by a set of weights and biases, and during the training stage, the model learns the parameters that together best predict the volume vectors in the train set, given the train features. This is done by backpropagation, the process of iteratively updating the weights and biases with a gradient descent algorithm. Once the model is learned, the training phase only requires the road segment features to be fed through the network to produce the 24-hour volume predictions.

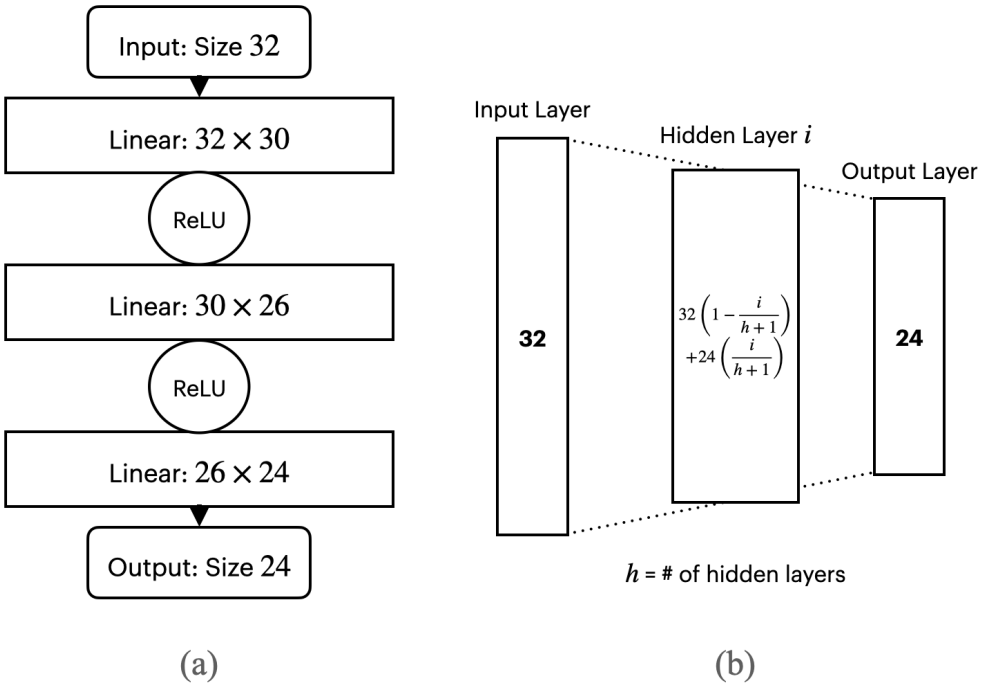


Figure 25: (a) Example of a neural network structure, (b) Calculation of hidden layer length.

Since deep neural networks have large functional capacity, models are very susceptible to overfitting if proper regularization methods are not implemented. Popular methods include weight decay and dropout, but for this project, we found *early-stopping* to be the most effective. Specifically, early-stopping is the process of terminating the iterative gradient descent algorithm when the test error meets a certain criterion for some consecutive amount of iterations. In our case, we check the test error every 100 epochs (iterations), and if the test error fails to improve

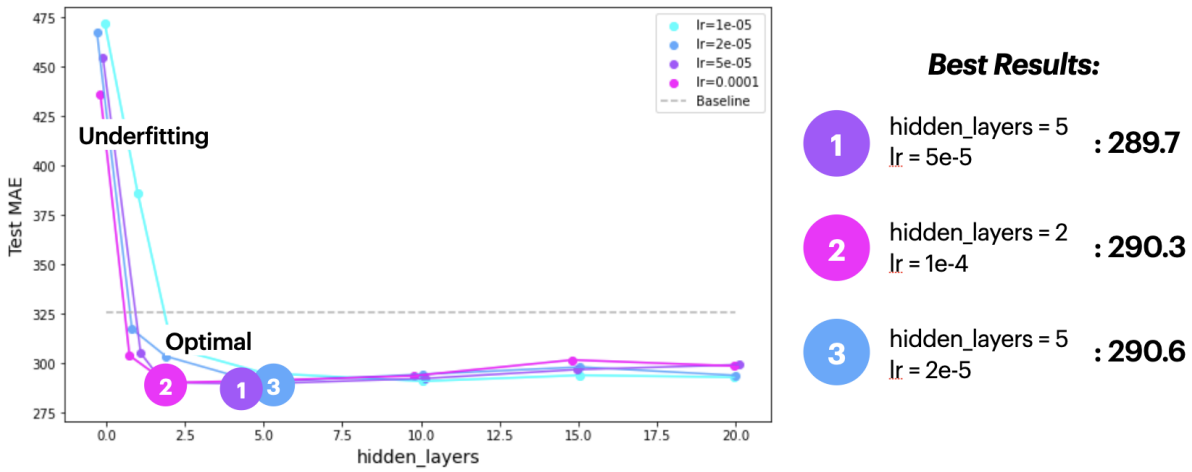


Figure 26: MAE results of hyperparameter tuning the neural network algorithm.

for 5 consecutive "checks", then we terminate the algorithm. Moreover, the neural network algorithm employs a variety of hyperparameters, two of which we tune in our experiments. The first hyperparameter is the *learning rate* (**learning_rate)**, which controls the rate of weight update during backpropagation; larger learning rates tend to converge quicker, but may not descend as low as the smaller learning rates. The second hyperparameter is the *number of hidden layers* (**hidden_layers**) in the network; this setting is affiliated with the model capacity, so more hidden layers will allow the model to learn complex patterns, at the expense of potentially overfitting by "memorizing" the train data. In our neural networks, the size of each hidden layer

is determined linearly, as shown in Figure 25(b). Since the inputs are length 32 and outputs are length 24, the ith hidden layer (out of h total hidden layers) has length $32(1 - \frac{i}{h+1}) + 24(\frac{i}{h+1})$.

In our neural network experiments, we hyperparameter tune on the values **learning_rate** $\in \{10^{-5}, 2 \times 10^{-5}, 5 \times 10^{-5}, 10^{-4}\}$ and **hidden_layers** $\in \{0, 1, 2, 5, 10, 15, 20\}$. In addition, we implement the algorithm with Pytorch's **nn.Module** class, including the MAE loss function, Adam optimization (Kingma and Ba), batch sizes of 32, and early-stopping (as described above). Figure 26 highlights the test performance of each hyperparameter setting in the search space; these numbers are also reported in Appendix A.4. Observing the graph, the tuning of **hidden_layers** $= 5$ appears to perform the best, while all of the **learning_rate** values perform interchangeably. These observations are the top three results, which all have a **hidden_layers** value $\in \{2, 5\}$, and varying **learning_rate** values. Precisely, the best-performing result yields a test MAE of 289.7, which scores 36.2 less than the baseline error and 2.5 less than the best random forest error. For **hidden_layers** $= 0$ (equivalent to linear regression), neural network models appear to *underfit* as the model capacity is too small to learn the patterns fundamental to the feature-target relationship. However, we notice that *no* hyperparameter combination produces an overfitting model because the applied regularization method, early-stopping, is not treated as a tunable hyperparameter in this experiment.

Following the previous three algorithms, we include snapshots of the traffic volume predictions from three neural network models at 3:00AM, 11:00AM, and 7:00PM in Table 6. The overfitting case reflects high **hidden_layers**, high **learning_rate**, and most importantly, *no early-stopping*; the optimal case reflects mid **hidden_layers** and mid **learning_rate**; and lastly, the underfitting case reflects low **hidden_layers** and low **learning_rate**. When early-stopping is

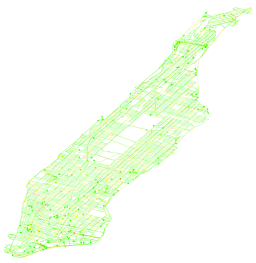
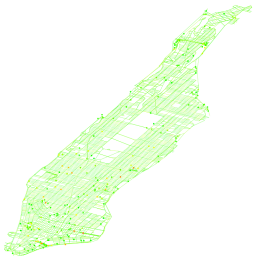
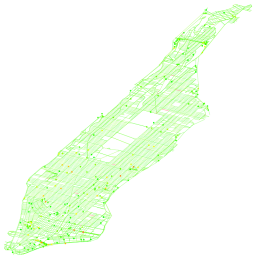
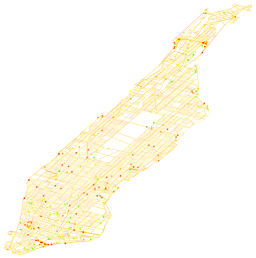
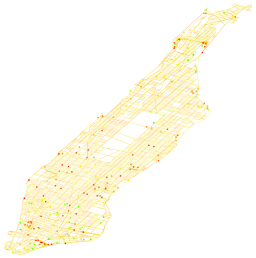
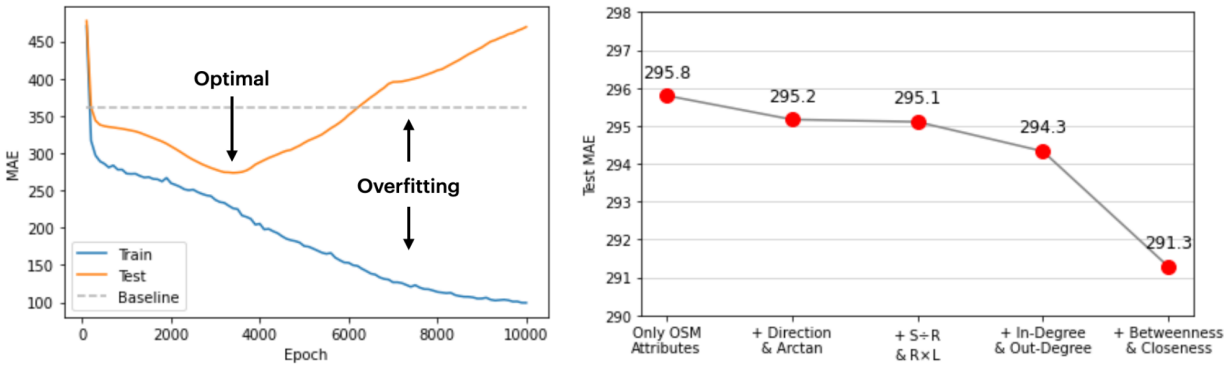
Case	*H*	3:00AM	11:00AM	7:00PM
Overfit	10 / 1e-4 ★			
Optimal	5 / 5e-5			
Underfit	0 / 1e-5			

Table 6: Visualization of neural network traffic volume predictions. Settings in the hyperparameter space *H* take the form: **hidden_layers** | **learning_rate**. ★: no early-stopping

left out, the model will train for a fixed number of epochs and potentially reach a point where the test error rapidly increases, while the train error continues to decrease. A prime example is shown in Figure 27(a): a model with high capacity (*e.g.,* 5 hidden layers) contains enough parameters to memorize the train data, which inevitably leads to extreme cases of overfitting. As

discussed before, the overfitting case reflects strong-colored predictions, while the underfitting case tends to promote uniformity across all road segment predictions. Diving deeper into the optimal case, feature importance may also be considered in the context of neural networks. Specifically, Figure 27(b) shows that adding each attribute group to the feature vector achieves improvement in test performance at each step; slight but consistent decreases in MAE are achieved by the first three additions of engineered attributes, while the final addition of betweenness and closeness centralities yields an improvement of 3 MAE, alone. Due to random seeding, the resulting test error of 291. 3 is slightly different than the previously-reported MAE of 289. 7 for the same setting; nevertheless, these results show that each attribute group contributes meaningfully to the learning of deep neural networks.



(a) (b)

Figure 27: Results given by a neural network with 5 hidden layers and 5×10^{-5} learning rate: (a) Train and test MAEs over time with no early-stopping, (b) Addition of each attribute group improves MAE.

4.7. Model Comparison

Now that the four ML algorithms have been detailed above, we can discuss the best-performing instances (with respect to the test MAE) of each algorithm. Specifically, we

identified three metrics that are important in evaluating the performance of a ML model: test error (MAE), train time, and test time. The first metric, *test error*, is a good measurement of the model's ability to generalize its performance to unforeseen examples. The next two metrics, *train time* and *test time*, measure how long the model needs to fit on the train data or predict the test data; we record these times by allowing each ML model to fit on the training examples (90% of the 310 SRS examples), then predict the rest of the ~9,600 road segments in Manhattan. Test time is particularly important because we prefer that our deployed models compute efficiently in real time. Table 7 lists each of the three discussed metrics for every ML algorithm's best-performing hyperparameterized instance.

	K-Nearest Neighbors	Decision Tree	Random Forest	Neural Network
Test error (mean ± std.)	309.0 ± 56.6	305.5 ± 69.0	292.2 ± 37.5	289.7 ± 48.7
Train time	0s	916ms	301ms	8.27s
Test time	295s	1.29s	1.92s	1.29s

Table 7: Model performance metrics across all four ML algorithms.

Analyzing the table, we first notice that the random forest and neural network algorithms yield much lower test MAEs than the KNN and decision tree algorithms; this is expected because the random forest and neural network models have much larger functional capacities, allowing them to learn more complex patterns in the relationship between road segment attributes and hourly traffic volume. Furthermore, it is not surprising that KNN gives the highest

test MAE because the algorithm is strictly concerned with the location of road segments in the network, thus ignoring other meaningful attributes (*e.g.,* speed limit and number of lanes). Next, we observe that the KNN algorithm requires no time to train– a characteristic of an instance-based learning algorithm, which stores each train example in memory for testing; Conversely, neural networks take the longest amount of time for training (8.27 seconds) because backpropagation with 5 hidden layers tends to be computationally expensive. Lastly, we notice that the KNN algorithm takes a significant amount of time for testing (295 seconds), primarily because the algorithm was implemented for this specific project and therefore very likely not computationally optimized. On the other hand, decision trees and neural networks both take much less time to test (1.29 seconds each) because the forward pass of the models are optimized by the Scikit-Learn and Pytorch libraries, respectively. Overall, the neural network algorithm appears to be the most appealing predictor of traffic volume because it yields the smallest test error and predicts the road segments in the fastest time.

5. MODEL EVALUATION & DISCUSSION

In the previous section, we explored the predictive capabilities of several standard ML algorithms with our generated and preprocessed dataset; each algorithm included a discussion on hyperparameter tuning, optimal results, and visualization of traffic volume predictions. Having discussed these technicalities and results, it is equally important to consider the bigger picture of our project. In doing so, we seek to address the following questions in the next four subsections:

- ★ **Model trustworthiness:** in this project, we fit models on a dataset of 310 road segments, but make predictions for 9,600+ unforeseen road segments. How reliable are these predictions? How do we know?

- ★ **Model performance:** we received generally similar results for each algorithm. What do these results mean? Can we really quantify "performance" with one metric? How does this affect the way models learn in our project?
- ★ **Project improvements:** how could we improve the outcome of the project? Should we go about the collection of data differently? With better data, should we keep the algorithms we use, or expand our consideration?
- ★ **Project merits:** we covered a lot of content in this project. What are the biggest takeaways? If appropriately refined, how could the deliverables of this project serve use to the public?

5.1. Model Trustworthiness

An underlying assumption in supervised learning is that training examples and testing examples are independent and *identically distributed* (Goodfellow et al. 109). The reasoning is simple: if a model performs well on a train set drawn from a joint distribution $Q_{X,Y}$, then the model should be able to generalize its predictive capability to *any* test set also drawn from $Q_{X,Y}$. In our experiments, we train models on data exclusively taken from the 310 stationed road segments (SRS's), then use the models to predict *all* ~9,600 road segments in Manhattan; the question is, can we assert with confidence that the train examples (SRS's) were reasonably drawn from the underlying Manhattan-road-segment distribution? If so, we can be confident in our models' ability to generalize to unforeseen road segments. However, if not, then we have reason to challenge the legitimacy of our models' predictions.

As mentioned in Section 3, the feature vector used in our model-based learning algorithms is composed of 36 values. Consider the feature vector as a 36-dimension random

variable: $F := (X_1, X_2, \dots, X_{36})$, where X_i is a random variable of the ith entry. If Q_F is the feature-generating distribution of SRS's (our training features), and P_F is the feature-generating distribution of *all* road segments in Manhattan, then we would like for Q_F and P_F to be as similar as possible. That way, we can train a model to learn traffic volume patterns conditioned on $Q_F \approx P_F$, then use the model to predict traffic volume for road segments drawn from the identical distribution. For further analysis, we must establish some notion of distribution similarity: a common metric in the ML community is the *Kullback-Leibler (KL) divergence* (Goodfellow et al. 72) defined below:

$$D_{KL}(P \,||\, Q) = \sum_{i=1}^{n} P[i] \times log\left(\frac{P[i]}{Q[i]}\right),$$

where P and Q are the ground-truth and estimated distributions, respectively. The metric clearly assumes that the sample space for each distribution is discrete, and of the same length (n); this poses a problem in computing $D_{KL}(P_F \,||\, Q_F)$ because F contains random variables (X_i's) that are continuous (*e.g.,* road length, arctan, betweenness centrality). Therefore, we must relax the complex random variable F to a discrete random variable B that preserves the distribution of road segments as well as possible. Specifically, we do this by partitioning key road segment attributes into several bins each, then taking the cartesian product of these bins to create the sample space for B. This process is shown by Figure 28 and 29(a), where 7 attributes are segmented into categories or intervals, then combined to form about 25,000 bins for B.

In doing this, our goal is for Q_B and P_B to be discrete approximations of Q_F and P_F, respectively; consequently, if Q_B and P_B are shown (not) to be similarly distributed, then we have reason to believe that Q_F and P_F are (not) similarly distributed. However, simply computing

Index	Attribute	Bins
0	GPS Coor. Cluster	0: {0}, 1: {1}, 2: {2}, 3: {3}, 4: {4}
1	Road Length	0: (–∞, μ – σ), 1: [μ – σ, μ + σ], 2: (μ + σ, ∞)
2	Speed Limit	0: {Missing}, 1: [0,25), 2: {25}, 3: (25, ∞)
3	# of Lanes	0: {Missing}, 1: {1,2}, 2: {3,4}, 3: [5,∞)
4	Direction	0: {NE}, 1: [NW), 2: {SE}, 3: (SW)
5	Betweenness	0: (–∞, μ – σ), 1: [μ – σ, μ + σ], 2: (μ + σ, ∞)
6	Closeness	0: (–∞, μ – σ), 1: [μ – σ, μ + σ], 2: (μ + σ, ∞)
7	Total Degrees	0: {1,2,3}, 1: {4,5}, 2: [6, ∞)

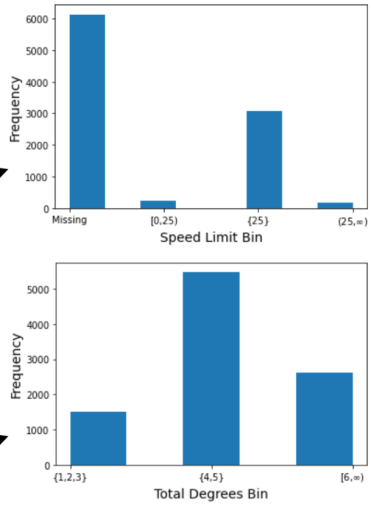


Figure 28: Partition of the feature space (left), along with distribution examples (right).

$D_{KL}(P_B \,||\, Q_B)$ is not enough to make this conclusion without proper context; therefore, we carefully develop a hypothesis test on the following question: is there any evidence to suggest that the train examples (SRS's) were *not* drawn from P_B, the (discrete) ground-truth road segment distribution? We begin with the formulation of a *null hypothesis*– the train examples were sampled from P_B– and an *alternate hypothesis*– the train examples were sampled from a different distribution; additionally, we set the significance level to be $a = 0.05$. Our approach is to draw 10,000 samples of size 310 (= number of SRS's) from P_B, and for each sample, compute $D_{KL}(P_B \,||\, \widehat{P}_B)$, where $\widehat{P}_B$ is the sample distribution over B. Figure 29(b) shows this histogram.

Having curated the data, we proceed with the hypothesis test. Assume toward a contradiction that the training examples were in fact sampled from P_B; that is, Q_B could have possibly been an instance of $\widehat{P}_B$ defined above. If Q_B were a sample distribution, then with

$100(1 - \alpha) = 95\%$ confidence, we would expect $D_{KL}(P_B \,||\, Q_B)$ to be less than Figure 28(b)'s upper bound, 2.76. However, we observe the *test statistic* to be $D_{KL}(P_B \,||\, Q_B) = 2.85$, which is not less than the upper bound: this contradicts our expectation with 95% confidence. Therefore, we *reject* the null hypothesis to claim the existence of sufficient evidence suggesting that training examples were *not* drawn from the ground-truth road segment distribution. In Figure 29(b), we compute the *p-value* to be 0.02, which is less than $\alpha = 0.05$ and consistent with our conclusion; however, if we had set the significance level to be $\alpha = 0.01$ prior to the execution of the hypothesis test, we could have easily declared the effect to be statistically insignificant. Ultimately, through this analysis, we find that the train and test feature distributions are potentially different enough to raise concern about our ML models' ability to generalize to unforeseen, non-SRS examples. These conclusions threaten the trustworthiness of our models and underscore the pressing need for data curators (such as NYSDOT) to sample road segments representative (with respect to the attributes) of the entire Manhattan network.

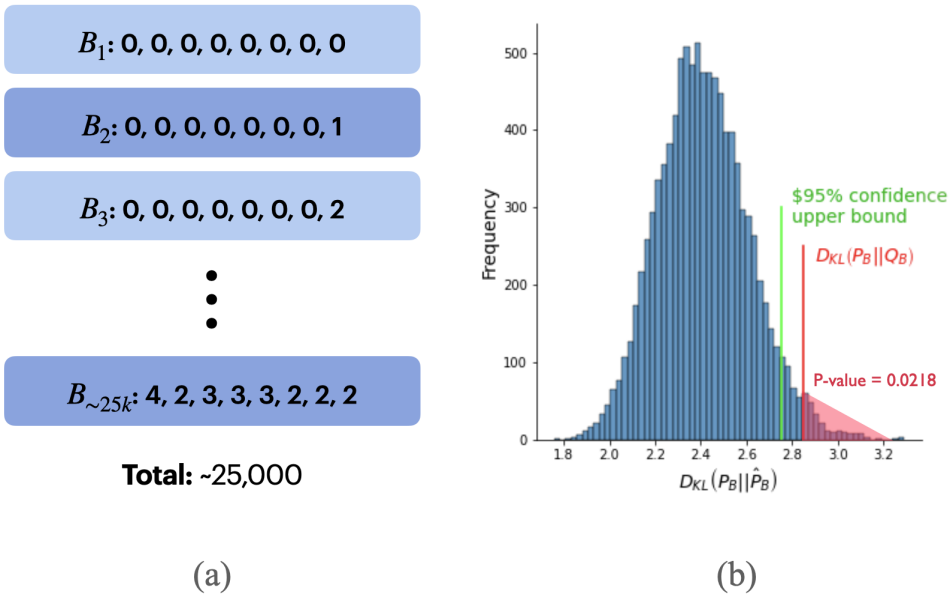
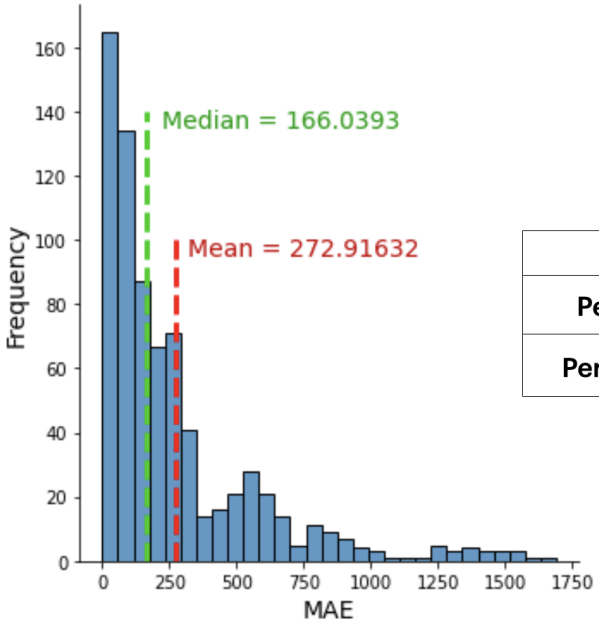


Figure 29: (a) Bins of segmented attributes via cartesian product, (b) Histogram of results.

5.2. Model Performance

In this subsection, we discuss the strengths and limitations of the way we quantify model performance– specifically, in regards to the MAE. For example, consider a neural network model with 5 hidden layers and a 5×10^{-5} learning rate: on the first fold (train-test split), this model achieves a test loss of 272.9. What does this value mean? On average, the model's *hourly* traffic volume prediction is off from the ground truth by 272.9 vehicles. This seems like a very large error, but if we consider the performance metric by every *minute*, the model's predicted volume count is only off by 4.55 vehicles. Similarly, every *10 seconds*, the model is only off 0.76 vehicles, which now appears to be a much more impressive result. Therefore, it is important to consider the context of the metric before evaluating the performance of an algorithm.

Moreover, recall that we define the MAE metric to be a mean across *all* road segments, so it is necessary to consider the whole distribution of *absolute errors*, which we define to be the



	25%	50%	75%
Per hour	67.6	166.0	332.0
Per minute	1.13	2.77	5.53

Figure 30: Distribution of absolute errors, along with quartile measures.

difference in volume between a ground-truth vector and predicted vector, for a given hour. Figure 30 plots the distribution of absolute errors achieved by the same neural network described above. Although the mean of the absolute errors (MAE) is 272.9, the distribution shows that the median is 166.0: this means that 50% of the predictions are less than 166.0 off from the ground truth, which comes out to 2.77 vehicles a minute. Furthermore, 25% of the time, the neural network predicts within 1.13 vehicles of the ground truth every minute, which seems to be very accurate. These results indicate that the mean absolute error is not always a perfect representation of the full distribution; this tends to occur when the data is significantly skewed. Although the MAE is particularly sensitive to outliers (with respect to the hourly absolute errors), this metric is typically necessary in deep learning models (*e.g.,* neural networks) because it is differentiable for backpropagation– as opposed to the *median* absolute error, which is not. However, it is always helpful to consider the full distribution of absolute errors when evaluating the performance of a model, since it gives a much broader perspective than just the mean.

5.3. Project Improvements

Having completed the project, we now seek to address the limitations and hold a brief discussion on ways to improve the results. The project can be split into two primary tasks– (1) the dataset generation process and (2) the implementation of ML algorithms– and both of which can be improved upon in any future work. We begin addressing the first task of dataset generation, which can be broken down into target-side and feature-side modifications. To recap, target-side modification primarily involves the curation of hourly traffic volume from the NYSDOT database, and the assignment of these vectors to road segments in the OSM network. Since we are taking the data from NYSDOT, there is not too much we can directly improve

ourselves, but nevertheless, some modifications in traffic volume collection would be helpful in deploying a better-performing model.

One modification would be for NYSDOT to *increase the number of traffic stations*, possibly from 310 to 500 (or even 1,000); in general, models tend to give more accurate predictions when learning on larger datasets, and a dataset size of 310 is certainly on the smaller

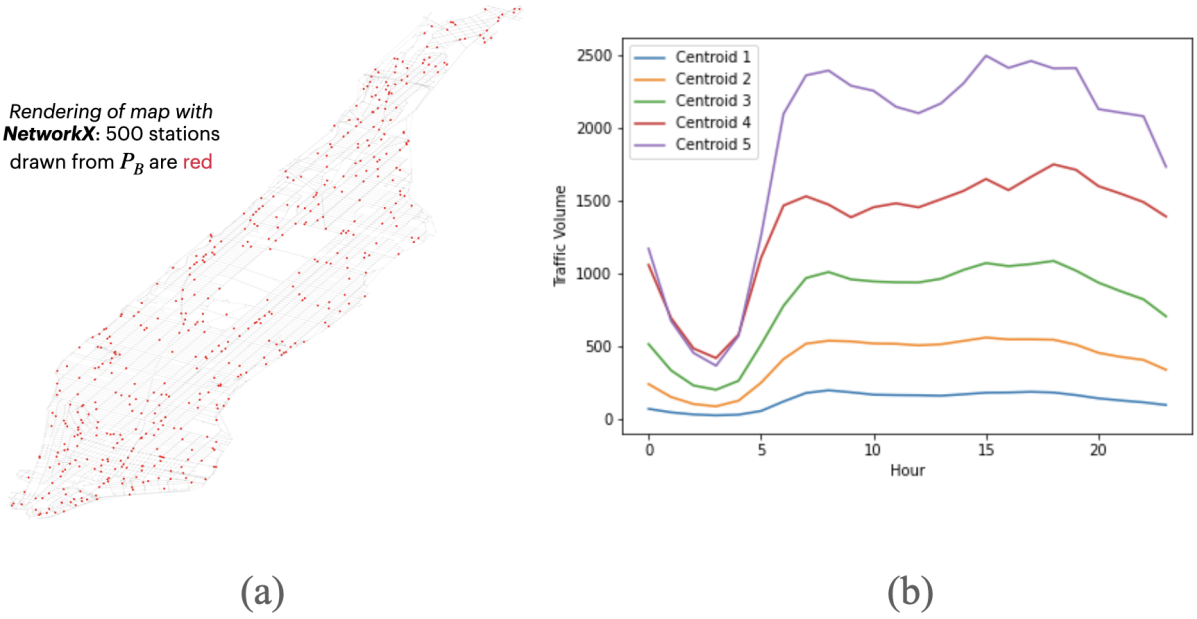


Figure 31: (a) 500 hypothetical traffic stations drawn from $P_{B'}$ (b) Example of 5-means clustering on the existing set of 24-hour volume vectors, reflecting a natural ordering of centroids.

end. Additionally, deep neural networks thrive on large datasets, so it would be convenient to reinforce our best-performing model (a 7-layer neural network) with more data coming from new traffic stations. Although improving the *quantity* of targets is a good first step, improving the *quality* of targets is equally important. In the "Model Trustworthiness" section, we conclude that the stationed road segments (SRS's) were unlikely to be drawn from the underlying Manhattan

road segment distribution; that is, the train examples were unlikely to be drawn from $P_B \approx P_F$ as defined above. Since B has a discrete sample space, it is possible to draw any number of road segments from P_B. In Figure 31(a), we draw 500 road segments from P_B and highlight their hypothetical traffic stations in red; if this were the new traffic station distribution, then we could ensure with high probability that the train examples (SRS's) and test examples (*all* road segments) are *identically distributed*, a fundamental assumption in ML theory.

Next, we consider potential modifications to the *feature side* of dataset generation; this primarily involves the selection and engineering of attributes from road segments. Clearly, the physical structure of each road segment (*e.g.,* length, speed limit, number of lanes, direction, *etc.*) was thoroughly embedded in its corresponding feature, but we believe more information about the surroundings of each road segment (*e.g.,* metrics on residence, work, and/or population) could have been more helpful in learning traffic volume patterns, had they been publicly available to us. One interesting example appeals to the betweenness centrality attribute; recall that the betweenness centrality of a directed edge (road segment) e is defined as:

$$C_B(e) = \sum_{s \neq t} \frac{\sigma(s,t,e)}{\sigma(s,t)},$$

where (s, t) is any pair of intersections in the network. The formula places equal weight on all possibilities of (s, t), but in reality, this should not be the case because some shortest paths between two intersections are certainly more commonly taken than others. Therefore, if we modify the betweenness centrality equation to be:

$$C_B^*(e) = \sum_{s \neq t} \lambda(s,t) \frac{\sigma(s,t,e)}{\sigma(s,t)},$$

where $\lambda(s, t)$ models the popularity of vehicles driving from s to t, then we derive an attribute that takes into account the relevance of each shortest path in the network. Likewise, the adjusted betweenness centrality could be normalized by setting the weight $\lambda(s, t)$ to $P(s, t)$, the probability that a vehicle begins at s and drives to t. This would serve to be a very useful metric, but the problem comes in the computation of λ; to our best knowledge, the data needed to quantify route popularity is publicly unavailable.

Next, we address the task of implementing ML algorithms that accurately predict traffic volume in Manhattan. Since our ML algorithms comprise a strong representative subset of supervised learning algorithms, we believe that modification to the data-generating process is the most important first step. If we improve data collection in terms of both quantity and quality, then better ML results will follow regardless of the ML algorithm used. However, we observe that train time and test time have the potential to be significantly reduced in future work by reformulating the ML task at hand. Specifically, from our analysis in Section 3.4, we notice that the 24-hour volume vectors can be easily represented by a single value– namely, the daily volume count. This suggests that if we perform K-Means clustering on the 24-hour volume vectors, then a natural ordering exists (from lowest to highest magnitude), as shown in Figure 31(b). Labeling the clusters $1 \ldots K$, we can reduce the task of multi-output regression to *single-output regression*, which would drastically reduce computation. For converting output to 24-hour volume predictions, a rounding to the nearest label is required, followed by a mapping from label to centroid (which is the final prediction). There would be less diversity in predictions (only K unique ones), but the accuracy should be preserved. As a result, dimensionality reduction is a promising avenue to improve the training and testing time of ML algorithms in future work.

5.4. Project Merits

For the sake of comprehensive evaluation, it is important to not only suggest potential avenues to improve upon in future work, but also highlight the positive takeaways of our project. In doing so, we consider the implications of this work through an *economic* lens. To put short, traffic station installation is expensive: in 2017, the New York State Department of Transportation purchased traffic stations for roughly $800 ("Traffic Volume Collection and Estimation on NonFederal Aid System Roads"). In this vein, the installation of 310 traffic stations would roughly cost $248 thousand, and the hypothetical installation of *all* road segments in Manhattan would roughly cost $7.7 million. Indeed, counting the traffic volume on each individual road segment in Manhattan would give us ground-truth results, but the installation is about 30 times more expensive than the mere installation of 310 stations. The difference in cost between the two scenarios underscores the *economic relevance* of a potentially successful project in this domain. Any ML algorithm yielding realistic and accurate traffic volume predictions from a small subset (~1/30th) of road segments would economically serve great use to state departments of transportation (DOTs). Although much improvement is needed in both the dataset generation and ML algorithm implementation stages, the nontrivial objective of cost efficiency is potentially a rewarding one for any successful future work.

(a) (b)

Figure 32: Installation of traffic stations by (a) Arizona DOT, (b) Wisconsin DOT.

6. CONCLUSION

In this work, we investigate the realistic generation of hourly traffic volume with machine learning (ML), using the ground-truth data of Manhattan road segments collected by the New York State Department of Transportation (NYSDOT). Specifically, we pose the following question– *can we develop a ML algorithm that generalizes the existing NYSDOT data to all road segments in Manhattan?*– and address the question by introducing a supervised learning task of multi-output regression, where road segment attributes are used by ML algorithms to predict hourly traffic volume vectors. In doing so, we consider four ML algorithms– K-Nearest Neighbors, Decision Tree, Random Forest, and Neural Network– and hyperparameter tune by evaluating the performances of each algorithm with 10-fold cross validation. Ultimately, we conclude that neural networks are the best-performing models and require the least amount of testing time. Lastly, we provide insight on model trustworthiness, model performance, project improvements, and project merits. Overall, we hope our project brings light to an interesting problem of cost efficiency, and opens new directions in the processes of supervised dataset generation and ML algorithm design.

7. REFERENCES

"Basic Network Analysis Tutorial." *Kaggle*, https://www.kaggle.com/code/usui113yst/basic-network-analysis-tutorial. Accessed 29 May 2022.

Evans, Jarrett. "Instance-Based Learning vs. Model-Based Learning." *Medium*, 29 June 2020, https://medium.com/@jevans2532/machine-learning-tales-44339a3c5711. Accessed 29 May 2022.

Goodfellow, Ian, et al. *Deep Learning*. MIT Press, 2016. Accessed 30 May 2022.

"HDSB Raw Traffic Data." *NYSDOT*, https://www.dot.ny.gov/divisions/engineering/technical-services/highway-data-services/hdsb. Accessed 29 May 2022.

"How Many People Use Google Maps Compared to Apple Maps? July 2021." *Justin O'Beirne*, https://www.justinobeirne.com/how-many-people-use-google-maps-compared-to-apple-maps. Accessed 28 May 2022.

"in_degree — NetworkX 1.10 documentation." *NetworkX*, https://networkx.org/documentation/networkx-1.10/reference/generated/networkx.DiGraph.in_degree.html. Accessed 29 May 2022.

Kingma, Diederik P., and Jimmy Ba. "Adam: A Method for Stochastic Optimization." *ArXiv*, 3rd International Conference for Learning Representations, 2015, https://doi.org/10.48550/arXiv.1412.6980. Accessed 29 May 2022.

"out_degree — NetworkX 1.10 documentation." *NetworkX*, https://networkx.org/documentation/networkx-1.10/reference/generated/networkx.DiGraph.out_degree.html. Accessed 29 May 2022.

Schwertman, Neil C. et al. “A Simple Noncalculus Proof That the Median Minimizes the Sum of the Absolute Deviations.” *The American Statistician* 44 (1990): 38-39.

“single_source_dijkstra — NetworkX 1.10 documentation.” *NetworkX*, https://networkx.org/documentation/networkx-1.10/reference/generated/networkx.algorithms.shortest_paths.weighted.single_source_dijkstra.html. Accessed 29 May 2022.

“sklearn.cluster.KMeans — scikit-learn 1.1.1 documentation.” *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html. Accessed 29 May 2022.

“sklearn.decomposition.PCA — scikit-learn 1.1.1 documentation.” *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html. Accessed 29 May 2022.

“sklearn.ensemble.RandomForestRegressor — scikit-learn 1.1.1 documentation.” *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html. Accessed 29 May 2022.

“sklearn.tree.DecisionTreeRegressor — scikit-learn 1.1.1 documentation.” *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html. Accessed 29 May 2022.

“Traffic Volume Collection and Estimation on NonFederal Aid System Roads.” *Federal Highway Administration Office of Safety*, New York State Department of Transportation, 21 March 2017, https://safety.fhwa.dot.gov/rsdp/downloads/fhwasa17034.pdf. Accessed 30 May 2022.

A. APPENDIX

In the Appendix, we include the quantitative results for each ML algorithm's hyperparameter tuning, beginning with K-Nearest Neighbors, followed by Decision Tree, Random Forest, and Neural Network. Each reported value is in the form "mean ± standard deviation", which is computed with respect to the 10 folds (train-test splits). The top three results are highlighted in green, blue, and purple, respectively.

A.1. K-Nearest Neighbor Results

$K = 1$	2	3	4	5	6	7	8	9	10
345.7 ± 38.9	339.3 ± 45.0	335.9 ± 59.2	324.1 ± 60.5	332.4 ± 67.9	321.8 ± 66.6	321.9 ± 64.8	316.3 ± 59.4	311.7 ± 54.3	309.0 ± 56.6
$K = 11$	12	13	14	15	16	17	18	19	20
310.6 ± 56.0	310.6 ± 56.2	312.2 ± 56.9	311.7 ± 59.4	311.6 ± 59.2	312.1 ± 59.6	314.9 ± 61.0	314.4 ± 63.3	316.0 ± 63.1	314.5 ± 63.8
$K = 25$	30	35	40	45	50	55	60	65	70
314.6 ± 64.6	312.1 ± 64.5	311.3 ± 64.2	312.1 ± 65.1	313.0 ± 65.2	313.3 ± 64.7	314.4 ± 64.1	313.3 ± 65.3	312.9 ± 65.1	314.0 ± 65.2
$K = 75$	80	85	90	95	100				
313.8 ± 65.0	313.6 ± 65.6	313.0 ± 65.5	313.7 ± 66.5	314.0 ± 66.6	314.7 ± 66.7				

A.2. Decision Tree Results

		max_features						
		0.4	0.5	0.6	0.7	0.8	0.9	1.0
max_ depth	2	320.7 ± 58.6	324.4 ± 56.8	321.7 ± 56.7	332.4 ± 56.0	332.5 ± 39.3	332.0 ± 49.0	316.0 ± 41.1
	3	318.9 ± 50.5	315.3 ± 39.9	324.7 ± 49.9	329.9 ± 52.1	328.8 ± 49.8	331.1 ± 62.9	316.7 ± 51.4
	4	319.2 ± 61.5	322.5 ± 40.8	325.4 ± 37.5	333.1 ± 55.7	330.3 ± 60.1	323.2 ± 61.6	322.6 ± 57.6
	5	333.4 ± 64.5	325.0 ± 54.2	326.3 ± 52.0	324.0 ± 57.2	325.2 ± 65.2	322.8 ± 57.2	326.3 ± 50.6
	6	330.7 ± 79.9	329.2 ± 55.4	321.9 ± 47.4	330.8 ± 50.1	331.9 ± 44.1	307.7 ± 45.6	323.8 ± 49.0
	7	335.3 ± 68.0	324.6 ± 60.0	337.9 ± 59.1	310.5 ± 60.5	326.8 ± 35.8	314.1 ± 56.6	330.6 ± 60.2
	8	333.1 ± 78.0	325.6 ± 54.4	330.0 ± 59.4	310.5 ± 68.7	333.3 ± 59.7	320.7 ± 69.0	328.1 ± 49.8
	9	330.8 ± 76.4	323.2 ± 66.4	334.6 ± 68.2	321.3 ± 72.4	335.2 ± 60.7	329.0 ± 57.7	332.1 ± 67.8
	10	339.1 ± 87.9	305.5 ± 69.0	345.3 ± 67.1	317.6 ± 78.7	324.3 ± 49.3	313.8 ± 64.1	337.6 ± 65.2
	11	319.8 ± 69.6	323.1 ± 64.6	340.3 ± 61.2	305.8 ± 68.1	333.2 ± 65.9	331.6 ± 57.7	329.7 ± 80.9
	12	367.6 ± 67.9	337.2 ± 78.0	339.2 ± 85.5	326.3 ± 78.9	353.4 ± 62.6	355.4 ± 78.8	336.2 ± 64.0
	13	358.8 ± 82.5	333.8 ± 74.6	340.2 ± 81.5	338.8 ± 71.1	328.2 ± 63.7	334.8 ± 58.9	356.5 ± 56.8
	14	368.3 ± 92.3	349.5 ± 104.8	352.4 ± 65.9	339.6 ± 76.9	367.0 ± 53.3	349.7 ± 66.9	352.1 ± 60.4

A.3. Random Forest Results

		max_depth				
		2	4	6	8	10
n_estimators	2	346.4 ± 47.7	338.8 ± 64.2	339.9 ± 41.3	372.8 ± 60.3	373.4 ± 58.1
	5	320.1 ± 47.0	312.0 ± 37.1	318.2 ± 38.8	324.5 ± 42.0	320.0 ± 30.5
	10	312.5 ± 47.2	298.8 ± 38.8	300.7 ± 43.1	305.0 ± 35.4	305.4 ± 39.7
	20	310.9 ± 47.5	301.5 ± 42.7	298.0 ± 38.4	298.5 ± 32.7	304.0 ± 35.4
	50	310.7 ± 43.5	303.9 ± 40.0	292.2 ± 37.5	297.3 ± 38.1	298.4 ± 43.9
	100	310.7 ± 44.4	300.3 ± 37.3	293.8 ± 38.1	297.3 ± 37.2	298.5 ± 37.4
	150	310.6 ± 43.6	297.4 ± 39.7	292.9 ± 37.0	297.6 ± 37.0	299.3 ± 35.3
	200	309.8 ± 41.3	299.7 ± 40.1	293.8 ± 37.6	295.9 ± 35.6	297.2 ± 40.1

A.4. Neural Network Results

		hidden_layers						
		0	1	2	5	10	15	20
learning_rate	1e-5	471.7 ± 61.2	386.2 ± 59.4	308.0 ± 57.7	295.2 ± 50.8	291.1 ± 48.2	294.1 ± 45.7	293.1 ± 44.6
	2e-5	467.3 ± 61.3	317.4 ± 57.4	303.7 ± 55.2	290.6 ± 49.3	294.6 ± 47.2	298.3 ± 45.3	294.0 ± 46.8
	5e-5	454.9 ± 61.3	305.2 ± 54.3	290.5 ± 50.6	289.7 ± 48.7	292.5 ± 46.4	297.0 ± 45.8	299.4 ± 48.0
	1e-4	436.2 ± 61.0	304.0 ± 55.4	290.3 ± 50.7	291.4 ± 49.2	293.8 ± 45.6	301.8 ± 50.8	298.7 ± 55.3