Clustering Algorithms to Further Enhance Predictable Situational Data in Vehicular Ad-Hoc Networks

by

Adam Samuel Dean

Farak Kandah
Professor of Computer Science
(Chair)

Anthony Skjellum
Professor and Director, SimCenter Computer Science and Engineering
(Committee Member)

Michael Ward
Professor of Computer Science
(Committee Member)

Yu Liang
Professor of Computer Science
(Committee Member)

Clustering Algorithms to Further Enhance Predictable Situational Data in Vehicular Ad-Hoc Networks

by

Adam Samuel Dean

A Thesis Submitted to the Faculty of the University of Tennesse at Chattanooga in Partial Fulfillment of the Requirements of the Degree of Master of Science: Computer Science

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

2020

ABSTRACT

The modern world is constantly in a state of technological revolution. Everyday some new technological idea, invention, or threat emerges. With modern computer software and hardware advancements, we have the emergence of more internet-enabled devices - or, Internet of Things (IoT) devices. We can now create networks with any device to gather real-time information. In conjunction, modern car companies have a push from public demand for a fully-autonomous car. In order to accomplish autonomy safely and effectively, Vehicular Ad-Hoc Networks (VANETs) must be established for a group of cars and their environment to ensure correct and relevant information is transmitted. The data collected in a VANET can be passed to machine learning models in order to predict conditions and detect anomalies. This thesis explores different ways of clustering groups of vehicles along with machine learning algorithms to predict where vehicles are likely to be and detect false or impossible information.

ACKNOWLEDGMENTS

First, I would like to thank my family and friends. You guys have always provided support for my various career and educational choices. More importantly, you have provided entertainment during times when I little pick me up went a long way.

I would also like to thank everyone with the SimCenter and IMSA. Corey and Jacob, you both provided me with a great learning experience and helped me learn some of these applications from the ground up. There are many people who have helped me in many different ways at the SimCenter, from getting door codes, to desktop support, to helping me find grants to get through school. Getting involved with the SimCenter and IMSA was one of the best decisions I have ever made.

I would like to thank all the committee members for this thesis as well. Dr. Skjellum, you were key in getting everything set up with IMSA which turned out to be a great project for all of us. You also always provide a good laugh. Dr. Ward and Dr. Liang, when I first came to UTC, I knew nothing about computer science, and both of you did an excellent job helping me learn many different fields over several short years. I appreciate both your patience.

Lastly but most importanly, I would like to thank Dr. Kandah. I could not have imagined being in this position walking into your office asking about a research job 2 years ago. I cannot thank you enough for the time, attention, education, and opportunity you have provided for me. You have changed my life in an extremely positive way.

There are many others out there, so thank you everyone who has been a part of my life thus far.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Over the past several decades in computing, two trends have reshaped what is possible in the world with technology: decreased size and increased power of computer hardware along with the increased ability for devices to have intranet and internet connectivity [3]. Moores Law, the 1965 observation of circuit boards being able to handle twice as many transistors every two years, has held true and the resulting hardware is visible in almost every branch of technology [3, 4]. A few examples are the progression of phones and cell phones, hard-disk drive density, network capacity, and even price of IT equipment [3]. Over time, we have (and will continue to be able to) dramatically increased our ability to design and access increasingly powerful devices at a lower cost. In conjunction, there has been a huge jump in demand for devices to be connected. We want to collect data from every possible source available to give us quick access to otherwise unknown information in our world. Our phones have slowly become small computers creating massive networks across the globe; cars and planes can now create networks along with providing a WiFi sub-net within itself; even trash cans can notify workers for emptying in densely populated areas [3, 5]. Networks have expanded immensely in both size and scope in a short period of human history.

Emerging from network expansion are Mobile Ad-Hoc Networks (MANETs), a network of moving (mobile) devices that are connected with the their peers in the network and ultimately back to a central data collection arena [6]. A subclass of MANETs is a Vehicular Ad-Hoc Network (VANET) created by semi-autonomous and autonomous vehicles. Cars are

now able to link together to pass information about road and traffic conditions in real-time. The whole network can know all node locations and movement statuses along with out-of-network conditions like road blocks or traffic lights. In these networks, huge chucks of data are constantly being passed between members of the network and an outside unit known as a Roadside Unit (RSU) and ultimately back to a main data store [7]. As the number of connected vehicles increases, the likelihood of cybersecurity threat decreasing network reliability also increases. Vehicular networks suffer from a wide range of unique challenges that are not present in other applications of cybersecurity [8]. Ensuring that VANETs and vehicle-to-vehicle communications are reliable, directly decreases the number of vehicle fatalities per year as well as costs associated with more minor traffic accidents. [9]. One unique cybersecurity threat to VANETs emerges from the ability for vehicles to share their location and movement information throughout the network. Malicious actors in the network can falsify movement information and create ghost vehicles in the network [10]. But false location information does not necessarily have to come from a bad actor. GPS failures, environment obstructions (mountains, buildings, etc), and network bandwidth can all result in loss of information [10, 11]. Essentially, without knowing a vehicle's location, or if it propagates a false location, can result in legitimate vehicles moving to an occupied position resulting in a collision. In addition, this type of attack can damage network routing, resource allocation, and previously trusted information [10]. Because of the likelihood of information error combined with high cost of human lives, a system to validate and predict location information for vehicles is absolutely required to keep Vehicular Ad-Hoc Networks safe and effective as they become more prevalent in modern society.

Modern machine learning algorithms have greatly advanced our ability to perform data analysis to gain actionable insight for human decisions [2]. The largest companies in the world use these types of algorithms for everything from product production to shipping tracking to customer analysis [2]. Each algorithm and algorithm type can provide further insight into data in different ways - a few are model selection, clustering, and regression.

We, in this work, propose a multi-layer approach to enhance predictions and validation of vehicle locations in VANETs. First, we cluster driving statistic into related groups to be used later in location predictions. Given a vehicular network, we can cluster vehicles based on their movement data, like direction, speed, and current location. Essentially, clustering will connect and relate all of our, otherwise unrelated, data points. The resulting grouping can be passed to a predictive model as an extra feature to enhance the accuracy in predicting future locations.

1.1 Research questions

This thesis will address the following research questions:

- Can machine learning algorithms aid in enhancing cybersecurity approaches in realtime environments? What combination of machine learning algorithms best supports a real life situation? What qualifies as an accurate model?
- Can the use of clustering further enhance location predictions?
- Can the use location predictions mitigate threats targeting vehicular networks?

1.2 Motivation and Contributions

With increasing demand for autonomous vehicles, the development of IoT networks and systems to process and act on data is crucial. Each vehicle will need to be able to make real-time decisions based on data fed to it from the network, and the outcome of the decisions will directly affect human safety [9,12–14]. Creating accurate models that are able to give vehicles information about other vehicles' likely locations is one of many ways to reduce on-road hazards along with the potential to mitigate threats in such networks. Our contributions of this work can be summarized as follows:

- **Construct of Multi-layer Machine Learning approach for VANETs**: By gathering vehicle data for a large portion of a city, we are able to construct a data set that any machine learning algorithm can be applied to. Our approach takes advantage of a common clean data set by using multiple algorithms in educated succession to deepen our understanding of the data, and ultimately, increase our ability to take action with the data set. The approach takes our data set and applies various clustering algorithms. The different algorithms will each classify a given data point into a group based on similarities determined by the algorithm. By doing so, similar data points will now be linked for later algorithms to benefit from. Conversely, non-linked data points will provide a mechanism for predictive algorithms to learn what is not related. Once the data is clustered, it can be passed through various machine learning regression models to predict target data points. Generally, machine learning regression models only predict a single target variable for given predictor variables. The vehicle location case examined in this paper involves predicting a future X coordinate and a future Y coordinate known as a multi-output regression case - fewer predictive algorithms support multi-output regression. The clustered data set will allow the regression algorithms to better predict both the future X and Y coordinates of a given vehicle. Thus, reducing risk in the system.

- **Demonstrate and evaluate the multi-layer approach to achieve the best model combination**:

  After creating several multi-layer models, we can compare each in several different ways to see which model best fits our use case. We can map our actual and predicted location values to see where vehicles are predicted to be against where they actually are in real-time. In addition, we can use more traditional model evaluations like average error and accuracy.

- **Evaluate the system under threat with deploying our multi-layer approach**: Once a higher accuracy predictive model is created, we can recreate scenarios that

emulate a false location threat. The clustering models will be able to determine whether or not a vehicle should be in a cluster or considered an outlier based on movement data. For example, a vehicle moving out of a busy intersection at a high rate of speed may no longer need to provide information to the network as it will soon be out of the network. The prediction models will be able to accurately determine whether or not a vehicle is in a plausible location. An example of this would be predicting that a vehicle should be moving across a street in a city, but instead, a few moments later, it is on the other side of the city. Our model will be able to validate and predict movements of vehicles in a network.

1.3 Organization

The remainder of this thesis is organized as follows: Chapter 2 will go over the background of the key tools used. We discuss related work in Chapter 3, followed by our observations that drive the work presented in this thesis. Our methodology is presented in Chapter 4, followed by the simulation setup in Chapter 5. We present our system evaluation in Chapter 6 and we conclude the work and present our future work in Chapter 7.

CHAPTER 2

Background

This chapter presents key tools in our research. Vehicle to Vehicle Communications are discussed in Section 2.1. Machine Learning and the algorithms used are presented in-depth in Section 2.2. Our threat model is examined in Section 2.3.

2.1 Vehicle to Vehicle Communication

The National Highway Traffic Safety Administration (NHTSA) estimates that 615,000 crashes, 250,000 injuries, and 1,400 fatalities could be prevented every year if a basic level of Vehicle to Vehicle Communication can be established [7, 9]. Some of the Basic Safety Messages (BSMs) defined by NHTSA are:

- Intersection Movement Assist - Alerts a driver when it is unsafe to enter an intersection
- Do Not Pass Warning - Alerts drivers about making dangerous passes
- Emergency Electronic Break Light Warning - Alert drivers in crowed areas that a likely non-visible vehicle has applied hard brakes
- Blind Spot Warning - Alerts drivers of potential obstructions or vehicles in the blind spot

Many of the BSMs presented by NHTSA are implemented or being implemented in vehicular networks. More importantly for our case, BSMs are based on and help create data

sets of vehicle speed, location, and direction information [9]. As vehicular network data becomes more advanced and readily available, we can use the information to create more complex models that are able to further prevent crashes, injuries, and deaths relative to the basic communications emerging today.

Figure 2.1 An example of how vehicular communications would appear in a real-world intersection [1]

2.1.1 Vehicular Ad-Hoc Networks

As mentioned, Vehicular Ad-Hoc Networks (VANETs) are a subset of Mobile Ad-Hoc Networks (MANETs) established from a relatively local group of autonomous vehicles communicating. There are a plethora of different ways of simulating VANETs. Some are extremely complex and detailed like Lyft's Level 5 project while are others are very basic simulators with few cars on a single lane road. For our case, we used Eclipse's Simulation of

Urban MObility (SUMO). SUMO creates models of traffic systems including road vehicles, public transit, pedestrians, traffic lights, emissions, traffic effect, and much more all in real-world environments. [15] In our case, we use SUMO's simulation of autonomous vehicles and platooning (groups of vehicles). From there, we can export simulation data to our own processing medium, Python, for further examination and analysis.

2.2 Machine Learning

Machine learning takes ideas from statistics, math, and computer science to create algorithms that can continuously learn on a data set to provide actionable insight. There are many areas of machine learning, but most are centered around some sort of preparation, prediction, and validation of data [2]. With that, almost every single industry has begun using machine learning for some sort of prediction - whether it be stock prices in financial markets or identifying better fields in agriculture [16]. This paper will examine clustering as a data preparation and classification step and use machine learning regression models to make predictions.

2.2.1 Clustering

Machine learning consists of two types of learning, supervised and unsupervised. Clustering is an unsupervised learning method meaning that training is done without any guidance. [2] Clustering is a useful tool for exploratory data analysis. The algorithm processes the data and groups it together without needing additional information about the data. There are several different methods used by algorithms to cluster data. One, density-based method group data points by how close they are to other data points. Hierarchical based methods use a tree-like structure in order to group data, and clusters are formed based on previously formed clusters. Lastly, Partitioning methods divide data with partitions and

each group will be its own cluster. [2]. This thesis will examine algorithms from each of these methods in order to best create predictions.

2.2.1.1 K-Means

First published in the early 1980s, K-means clustering rose to become one of the most popular partitioning clustering algorithms [17, 18]. Using Euclidean distances, it partitions data-sets into clusters by computing the nearest average of each neighbor [2, 19]. Euclidean distances can be defined as the direct distance between to points; it is very similar to the Pythagorean theorem distance of a hypotenuse. However, Euclidean distance find similarity between numbers that are not considered true X and Y plane distances. Basically, Euclidean distances can be calculated for any type of numeric variable such temperature, price, age, and all other numeric values. Euclidean distance also supports multiple dimensions meaning it can compare multiple variables of data (not just X and Y distance like Pythagorean). The algorithm will iterate through until a convergence on a predefined number of clusters [17]. Overtime, K-Means has maintained its popularity for several different reasons. Its speed and ease of implementation in practice are rarely beat [18]. In addition, K-Means has scaled easily as data sets have grown exponentially since its inception in the late 1970s [18]. K-means has no trouble handling the significantly larger data-sets. With uniform handling of data-sets, K-means has also remained a great tool acting as a common baseline for more modern clustering algorithms [19]. The ability to handle massively varying data-set sizes and baseline use-case notoriety are two key reasons we chose K-Means as the first clustering algorithm to implement.

2.2.1.2 BIRCH Clustering

BIRCH Clustering (Balanced Iterative reducing and Clustering using Hierarchies) is a hierarchical clustering algorithm known for its ability to handle large data sets with multi-dimensional data [20, 21]. Hierarchical clustering algorithms cluster data into hierarchies from a single cluster to an individual cluster for each data point [22]. Although we are limited by memory, BIRCH was selected because of its success with large data sets. In real-world vehicle networks, massive amounts of data will be moving through a system at any given time. In addition, BIRCH can begin iterating and learning (and thus clustering) without the full data set [20].

2.2.1.3 DBSCAN Clustering

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a clustering algorithm that builds clusters based on density of data points. High density regions, where data points are close together, will be grouped with their very close neighbors [2, 23]. On the contrary, lower density regions classify data points as outliers (also known as anomalies). DBSCAN has become one of the most popular algorithms for anomaly detection [23,24]. DBSCAN was chosen to be examined in this paper due to its success with anomaly detection, minimal parameters and data preparation, and low computational time and memory cost [23].

2.2.2 Machine Learning Predictions via Regression Models

Machine learning is a set of computer algorithms based on mathematical models that can continuously improve through iteration. Many different algorithms have been have been designed and implemented to solve a wide range of problems. From financial market analysis

to speech recognition, machine learning models dramatically enhanced how we use data to solve problems. One relatively common application is predicting a numerical value (target variable) for a given data set. Example applications include companies predicting when an employee might retire, predicting likelihood of child abuse based on at-home factors, or stock value predictions. Extending from single target variable predictions, some machine learning algorithms support multi-output regression. In other words, algorithms can predict multiple values for a given data set. This thesis will examine and compare a few of these algorithms to find which ones best predict an X and Y location of vehicles based on VANET data.

2.2.3 Multiple Output Regression Machine Learning Models

Oftentimes in machine learning implementations, the goal is to predict a single variable known as a Target Variable [2]. However, some cases require prediction of 2 or more variables. Sklearn provides a few methods to accomplish this - Multiclass Classification for non-numerical variables and Multioutput Regression for numeric variables [2]. Our case solves for both X and Y positions of a given vehicle. We stack our multiple output models on top of clustering models to create a more accurate prediction of X and Y values.

2.2.3.1 Multiple Linear Regression

Multiple Linear Regression acts as a linear predictor for multiple possible target variables [2]. Sklearn uses Ordinary Least Squares to calculate linear predictions and defines them mathematically as:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + ... + w_p x_p \tag{2.1}$$

Where $\hat{y}$ is your prediction, w, is the intercepts and coefficients of each predicted x value [2]. Multiple Output Linear Regression was chosen because of its wide use and

high ranking across many different scientific disciplines [25]. In addition, linear regression provides a simple trend line - a line visually correlating data. The trend line acts as a great base model for more complex model implementations [25].

2.2.3.2 KNeighborsRegressor

KNeighborsRegressor examines and predicts data points based on the distance to it's neighbors [2]. Essentially, it makes predictions based on closest known values - or, neighbors. KNeighborsRegressor performs strongly in situations where target data is both continuous and numeric [2]. Weights and neighbors preferences can be added to better fit data sets. Since our case is case is predicting future possible X and Y values of a given vehicle, KNeighborsRegressor is a strong model for us to implement for comparison. We used 5 neighbors with uniform weighting.

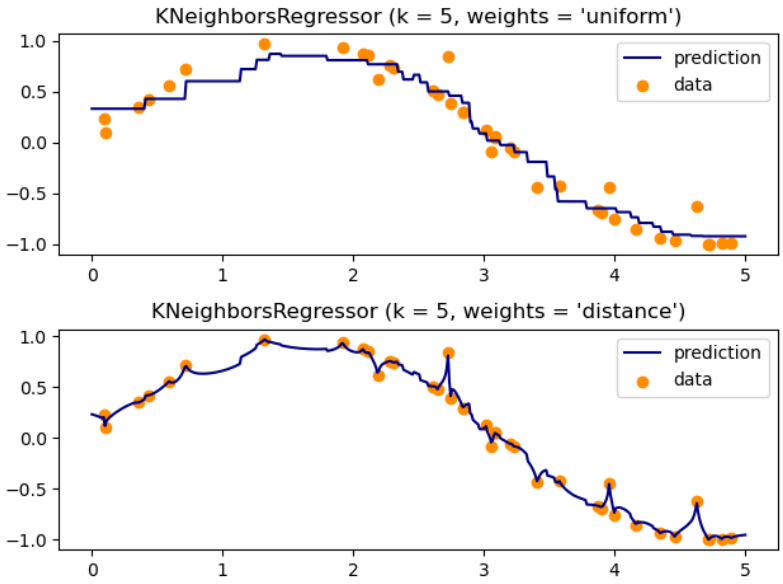


Figure 2.2 An example of how KNeighborsRegressor predicts data points [2]

2.2.3.3 DecisionTreeRegressor

DecisionTreeRegressor is a supervised learning method that continuously learns from features in the full data set [2]. Decision Tree's support for multiple output and supervised nature are the reason for implementation selection. In addition, DecisionTreeRegressor requires no additional data preparation (dummy variables, data normalisation, etc) like other supervised learning algorithms [2].

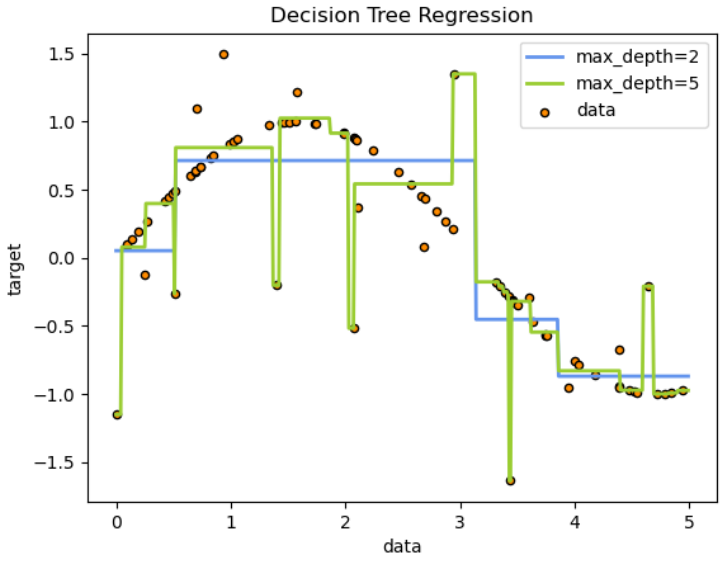


Figure 2.3 Example of Decision Tree prediction with different size trees [2]

2.2.4 Cross-Validation

We will stack each of these prediction algorithms on top of our data clustering to yield the accurate possible prediction for vehicle position determined by lowest Mean Absolute Error - the standard for accuracy in Sklearn regression models [2]. We ensure the accuracy of our models using a cross-validation strategy. Cross-validation is a strategy that partitions data into testing and training samples then running test data into multiple samples of iterating training data to validate [2]. Each of our clustering prediction models will

be validated with a 10 split, 3 iteration cross-validation function.

2.3 Threat model

Vehicular Ad-Hoc Networks face a plethora of cyber-security related threats. Threats can range from falsifying trust-worthy data to sending false signals about road conditions to falsely reported location data [7]. Our predictive models deal primarily with vehicle locations, so the key threat model we will examine and evaluate in this paper are replica attacks and injection attacks.

2.3.1 Replica Attacks

Replica attacks occur when a vehicle appears in two different locations at the same time [10]. This can occur from many different sources - a malicious actor could be copying a vehicle's information, or a hardware, like GPS, error could occur claiming a vehicle was not where it was actually. Either way, replica attacks can cause network resources (bandwidth and routing) to be mis-allocated, and thus, compromising network reliability [10, 11]. This can cause loss of data throughout the network, and ultimately place human lives in danger if vehicle locations are suddenly incorrect. By examining replica attacks as a threat, we will be able to mitigate the risk using our predictive clustered models. We can continuously evaluate data as vehicles move through time and the network to verify that threats are not present. Our use-case evaluation will be able to examine vehicles predicted location and verify that location, at least within our predicted accuracy, is in fact correct.

2.3.2 Injection Attacks

Injection attack occur when a vehicle's information is lost for a period of time, and then the vehicle is 'injected' back into the network. These types of attacks can be somewhat difficult to detect because there has to be some sort of tolerance for completely new vehicles entering a network [26]. Using our multi-layer predictive model, we will be able to identify and accurately account for lost data of vehicles' locations.

CHAPTER 3

Related Work

This chapter will examine previous work done in the field of VANETs and machine learning. Section 3.1 will cover machine learning and clustering. Section 3.2 examines data classification techniques. Section 3.3 looks at different malicious attacks and detection. Lastly, Section 3.4 goes over observations of related work and states the problem that will be addressed in this paper.

3.1 Machine Learning and Clustering

There is quite a broad range of research done on clustering in networks as a whole from basic small static device network clusters used for environment measurement to large complex moving networks. Most research into VANET clustering is focused on broadcast reliability and protocol, information ranking and reliability, and vehicle location.

Slavik and Mahgoub in [14] proposed a broadcast algorithm to maximize coverage efficiency in for Vehicular Ad-Hoc Networks. The authors implemented a distance-to-mean method in order to solve the broadcast storm problem which occurs when a VANET is overflooded with traffic resulting in poor performance becasuse of packet collisions and congestion.

In a given VANET, IEEE 802.11p allows for up to a 1000m transmission range, and there can be up to 80 cars with 20m of each other in a 200m radius. Thus, these networks

can vary greatly in density. Using multi-hop broadcasting protocol, they are able to handle many different situations to maximize network coverage for a given density. Distance-to-mean is based on the case where nodes should rebroadcast when relatively little of their transmission range is covered by other nodes. It works by estimating coverage based on the distance from a specific node to all other nodes in the cluster. If the average distance is very large, meaning that the cluster has spread apart, the node should rebroadcast to find a new cluster. The idea is to find a optimal threshold of distance for rebroadcasting.

The authors used the MLP (Multi Layer Perceptron) [27], a method used to approximate any function's arbitrary's accuracy. The authors are then able to create optimization algorithms from open source software packages - the algorithms compared are: Covariance Matrix Adaption Evolution Strategy [28], Particle Swam Optimization [29], and a genetic algorithm call Galileo [30]. To show the advantage of the Distance-to-mean broadcast protocol, the authors compared their approach to two other well known propogation protocols known as Advanced Adaptive Gossiping (AGG) [31] and p-persistence [32]. With that comparison it was proven that distance-to-mean protocol provided higher combined value for both reachability and bandwidth.

From the literature review, we observed that previous work is too focused on network coverage and bandwidth. Although important, the authors only use machine learning to find an optimal threshold value. They do not use algorithms to cluster nodes or predict any outcomes. The focus is too heavy on optimal node spacing and coverage.

A number of clustering approaches were based on entities' position to determine the optimal clustering [33, 34]. The authors in [33], applied machine learning to maximize coverage in drone network. Drones are pre-positioned around a city or given area with the goal of provided the strongest amount of wireless communication coverage over the biggest area possible. Drones and users can both move and cluster to accomplish this. The author's solution tackles the coverage problem when users and drones are both moving in a 3D environment and predicting mobility of users' to better support the network. Their

system is a two-faceted approach. First, a Q-learning algorithm is used that rewards a strong initial position (deployment), which is is given for any improvement in sum rate that will eventually converge to an optimal solution, at least until new user movement [33, 35]. The second, predicts user movements. The author's used, an Echo State Network, a type of neural network, to predict user movement [36]. To solve for user movement predictions, the authors examined a neuron reservoir size (number of neurons in the neural network), radius, and sparsity of that network to predict a user's location as a matrix representation ($\boldsymbol{W}$). The goal is to find the optimal number of neurons in the reservoir to minimize the Mean Squared Error of the $\boldsymbol{W}$ prediction. Ultimately, they find that using a neuron reservoir size of *n=1000* with their Echo State Network algorithm has a lower Mean Squared Error than other common Long Short Term Memory and Historical Average models [33].

In the case of drone position, 2 different algorithm models are implemented. One, a single agent model where drones do not give and receive information in the network. They use the single agent model more as a base example of how the second model is derived and valid [33]. Second, they create a multi-agent model where drones are all communicating with each other. Q-learning requires a Q-table to keep track of reward in a model [35]. Drones are only required to keep a Q-table of their own actions in the single agent model. However, in the multi agent model, each agent's Q-table includes data about both it's own states and actions as well as others. The goal is to maximize the reward function using the data from the Q-tables. The researchers go about maximizing the reward by individually training the Q-model for each agent [33]. Eventually, each Q-table will find an optimal reward value based on all agent actions - a successful model will have all Q-tables converge on a reward value continuous through $time = k + 1$ where k is the current unit of time [35].They found that after about 4000 learning episodes, the Q-tables would converge on an optimal throughput value based on learning rate. More importantly, they found that although higher learning rates converge faster, it can lead to a sub-optimal Q and thus lower throughput.

This work creates models on two separate agent sets: users and drones. However, it

only predicts movement of one set (users) and only creates a reward system for one (drones). A more accurate model could be created through predicting and rewarding a set of agents. If we can correctly predict where a user is moving and reward him (or our model) for predicting correctly, we can create a faster optimal throughput convergence model based on correctly predicted actions.

3.2 Data Classification

In addition to optimizing clusters based on location and communication to insure reliability and throughput, additional research has attempted to classify communications through VANET information ranking. VANET nodes receive many different types of information from many sources like speed and location from other vehicles, RSU signals, traffic information, communication from outside the cluster, and others. In the case of a VANET, transmission range and bandwidth limits the total amount of information that can be transmitted and stored on a network [34]. The authors use machine learning classification algorithm to find the most relevant information for each node in a network. The author creates a model to rank information using node reports that can be stored in a limited size per node database. When the database is full, re-ranking occurs and lowest ranking reports are discarded. Relevancy is determined by a Boolean (0 or 1, good or bad) approach based on nodes of the same cluster; known as feedback nodes in their case.In their model, reports are ranked by age and distance. Distance, in this case, is the distance from the vehicle to location of which the information refers. A simple would be a report about an empty parking space. If the report is new and receiving vehicle is close, the car can easily find an open spot, and the information can be deemed good. On other hand, a car far away receiving information later will not be able to get to the spot in time, thus the information is bad [34]. The goal is to rank each report based on likelihood the message is new and relevance of the message determined through neighbor relevance. This is done through a machine learning algorithm

that determines newness based on age and distance. Then a Naive Bayes learning maps the attributes to the probability. They create two example goals: parking spot availability and travel time prediction. Newness is derived from a duplication model - if a message is not new (or unique), it is likely to be older. A negative or positive value will be assigned, then, the Naive Bayes learning can aggregate results to learn an optimal convergence. By comparing rankings base on their algorithm, age, distance and a known Information Guided Search algorithm, they are able to prove that their algorithm outperforms just age and distance calculations as well as matching the known IGS [34]. For travel time, they compare to a known TrafficInfo algorithm. Outside of the distance parameter, their algorithm consistently had lower travel time than the known algorithm. Thus, combined with comparable results in the parking space simulaltion, their created algorithm proved to be more efficient in disseminating useful relevant messages.

A limitation in this paper is that it only compares their algorithm to one other for each case. This solution is too niche to just a parking and travel time simulation relative to one other solution. To find real information, a given solution should out perform multiple solutions on multiple cases.

The authors of Data Congestion implement an algorithm to solve the key problem of overcrowding in VANETs â i.e. traffic [37]. Red lights being the main congestion area. They does this through K-means clustering, an unsupervised learning technique. The authors introduces us to various congestion control techniques like rate-based, power-based, scheduling-based, and default congestion control strategies. Using varying values for transmission range, rate, window size, and delay, [37] is able to compare known control strategies to their own closed-loop hybrid strategy. In this case, closed-loop means that congestion is only handled after detection; hybrid refers to combining multiple known strategies. They create an environment with a congestion detection unit, a data control unit, and a congestion control unit with the goal minimizing average delay and packet loss while maximizing throughput and thus reliability. The congestion detection unit is tasked with recognizing

congestion based on message queue, channel usage, and channel occupancy time. Once a congestion is detected, the data control unit implements a K-meanings clustering algorithm to classify relevant data flowing in the network in order to reduce traffic on the network. Now that the clusters have been determined, the congestion control unit will analyze available clusters and adjust the transmission rate, range, and size to maximize performance (low delay, high throughput. First, the number of K-means clusters are determined by examining delay and packet loss relative to number of clusters. The features used are validity of messages, message size, distance between vehicles and RSUs, type of message, and direction of vehicle. Four is ultimately the recommended number of clusters. They show their algorithm to be superior to other singularly focused algorithms like the rate-based, power-based, scheduling based, or default [37]. The packet loss and delay are lower with higher throughput in all scenarios with varying parameters.

The key problem with this paper is the create their own multi-step scenario with different units to maximize their algorithms performance. This paper, along with others examined, do not contain any sort of anomaly detection. None of the papers to this point have examined anything related to malicious messages or misbehaving cars.

3.3 Malicious Actor Detection

One work presents an early version of determining network traffic in a cluster as normal or intrusive [13]. The authors classify intrusion into two separate approaches: misuse detection and anomaly detection with their intent to focus on anomaly detection in wireless networks. The paper presents a K-means clustering approach to classify network activity (as normal or intrusive) based on cluster radii, cluster size, and total network size to determine an ideal segragation point for *normal* or *intrusive* messages. The theory that they propose is that much more normal activity will occur than intrusive. Therefore, larger clusters will mostly be comprised of normal messages. Thus, they can create a model of normal attributes

from larger clusters and test for intrusision on smaller more distant clusters [13]. To go about classifying, a K-means clustering algorithm is implemented. K-means is used because of it's widely known success identifying coherent groups based on similarity of there attributes [13] Once the clusters are determined, the largest clusters is deemed *normal* and will be used as the base case. Then, looking at the smaller clusters, based on the base case, they compare the Euclidean Distance to find anomalies or intrusions. The goal is to minimize the false positive rate, meaning the predicted values of intrusion and normal are accurate. Otherwise, re-cluster. Eventually, there will be an optimal number of clusters for which predictions will be most accurate. After a learning and enhancement, they ultimately achieve a false positive rate of *fpr=0.15%* meaning that *99.85%* of their clustering model classified correctly [13].

This paper is too simple, mainly because it is older. It only implements uses one repeat K-means classifier to prove an optimal number of clusters. Now, the concept they implement is widely used. They also only deal with static wireless networks, and there is no account for vehicles or node movement. It is simply a grouping, a test, and a regrouping. Anomaly detection is much more advanced today, and many more cases can be explored outside of a simple binary *yes or no* based on static location.

A more modern paper, [12], examines security threats in VANETs. Like the intrusion paper, misbehavior detection implements a binary classification for security. However, authors add in additional parameters (other than, but including location) to classify nodes as either *honest* or *misbehaving*. To demonstrate effectiveness, the authors create a series of possible attack models.

- Packet Suppression Attack - Safety messages are not forwarded to neighbors
- Packet Replay Attack - Normal data is fraudulently repeated
- Packet Detention Attack - Normal packets are delayed
- Identity Spoofing Attack - Node pretend to be neighbors

- Position Forging Attack - Information refers to the wrong location

The goal is to create an algorithm to detect possible attacks using machine learning classification. To tackle this problem, the researchers compare Naive Bayes, IBK, AdaBoost, J-48, and Random Forest classifiers across the following features: position, RSU Range, Speed, and Recieved Signal Strength. From there, the authors examine results based on packet transmissions and drops. To find the best classifier, they compares based on True Positive Rate (TPR) and False Positive Rate (FPR) [12]. A true positive occurs when a malicious node is correctly identified. A false positive (also referred to as a false alarm) occurs when a legitimate vehicle is deemed malicious. The authors find Random Forest and J-48 classifiers are able to perfectly identify malicious actions in the network.

The key issue with this paper is they did not use a real scenario. They authors explain that the simulation parameters used are very specific [13]. They only use a 2-lane highway scenario with a set number of vehicles with fixed transmission range. In reality, VANETs are very fluid networks. Cars are constantly randomly moving in unfixed locations. Algorithms can too easily learn in their case, it would be better to classify in a more realistic scenario.

3.4 Observations

From the literature review, we observed that previous work can be too focused on some areas of studies while others can observe too broad of a scenario and miss detail on key points. For example, Slavik and Mahgoub [14] focus too much on network reach-ability and coverage. The distance-to-mean method's only goal is to find optimal coverage based solely on current location. Although broadcast grouping is similar, there is no classification of clusters to help identify where rebroadcasts may occur. A clustering strategy will help predict when a rebroadcast will happen based on location and reduce time to optimal coverage. The drone based paper also presented too niche of a scenario [33]. The situation looks at groups

of drones and users where only users are moving. In addition, drones receive a reward for optimal location (where users do not), and users movements are predicted (where drones are not). A simulation where correctly predicted movements will be rewarded will create better trust in the network - rewards could also be extended to providing positive messages to further increase trust.

One paper examines messages transmitted in a VANET and classifies them to improve network communications [34]. However, the authors only looked at two specific scenarios: parking availability and basic travel time. In addition, they only used two parameters, age and distance, to create their algorithm which is only compared to one other known. A more real world scenario would be examine all messages moving through a VANET based on time, sending and receiving vehicle position and speed, and source and destination cluster. By looking at more messages overall, you will be able to better classify messages as good or bad, increase network trust, and better detect anomalies.

Some authors use more features to determine clusters than any other paper [37]. From there, they are able to optimize delay and throughput relative to known algorithms that focus on few features [37]. One feature they use is message validity - there are conditions provide of what determines a valid message. Therefore, there is no way of determining good and bad nodes. This paper lacks a safe fall for anomalies and misbehaving nodes in the network. The older K-Means based paper [13] and the misbehavior detection [12] paper provide a solution for detection of misbehavior and anomalies. At the time of writing in 2005, K-means intrusion [13] was probably very advanced. It provided two machine learning algorithms that were able to classify messages based on basic parameters. Although useful still and provides a strong case for K-means clustering, the concept is basic relative to today's standard of identifying misbehavior in a network. For example, the authors of the misbehavior detection paper [12] presents multiple different attack models for a given network. With those models, the author's look at several different machine learning algorithms that are able to detect anomalies and thus attacks on the network. Ultimately, the authors are able to identify

misbehavior more accurately in a more advanced scenario than the K-Means based paper with newer techniques. However, the authors also state that their scenario was unrealistic [12]. They only used a 2-lane highway scenario with vehicles entering and exiting at a fixed rate. In reality, VANETs are very fluid networks, cars are constantly moving in unfixed environments.

3.4.1 Problem Statement

There are many different ways to cluster a given data set; different algorithms will perform better in different situations. After clustering a data set, an additional cluster variable is created for each data point that can enhance the accuracy of a later prediction from the added feature. The first part of this thesis will identify the best clustering algorithm and prediction model combination based on lowest average error of prediction. We can then apply the best modelâs predictions to our threat cases to identify and mitigate risks associated.

CHAPTER 4

Methodology

Our approach consists of a set of steps that are presented in Figure 4.1. This chapter will discuss each step as it relates to our case. Section 4.1 explains data collection. Section 4.2 shows the benefits of clustering, and Section 4.3 goes over our machine learning prediction models. We close with a concept of operations in Section 4.4.

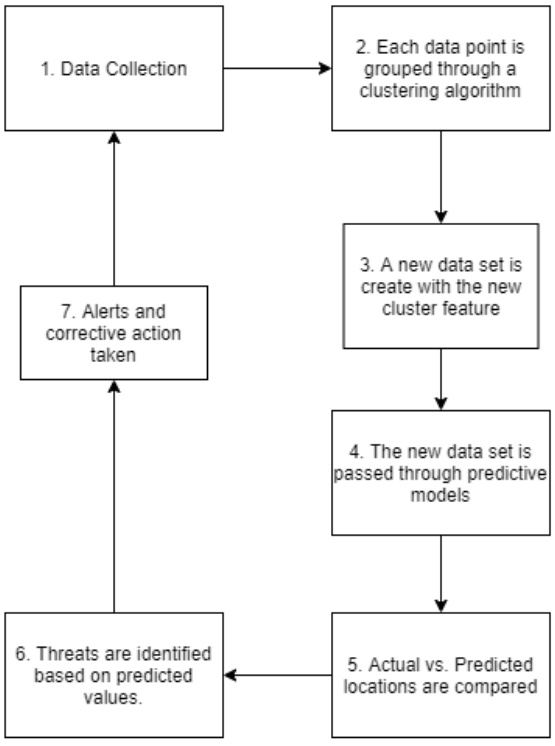


Figure 4.1 Data is collected and passed to a clustering algorithm. Clustering will create new feature on our data that will enhance predictions. The predictions can be used to decrease risk

4.1 Data Collection

The first item required to create safer road environments is gathering real-time data about the vehicles moving through a desired area. Each vehicle is solely responsible for collecting and propagating their own driving information. Chapter 5 will explain more about the simulation and data. But, a brief overview of the movement information that will be collected and processed are as follows:

- Time - The current time
- Direction - The direction of vehicle travel
- Identifier - A unique ID of each vehicle
- Position - A more local position variable. This could represent position in an intersection or lane.
- Speed - Current speed of the vehicle
- Location - A coordinate representation of the vehicle location

4.2 Clustering

From these variables, we can use clustering to group vehicles based on the collected variables like time, direction, and speed. Different clustering algorithms will group data points in different ways based on how the algorithm calculates clusters - we discuss this further in Chapter 5. Clustering will create an additional feature based off all of our data that will enhance our predictions later. Clustering will help predictions by linking similar data points together that the algorithms can use to help learning and predictions. On the contrary, non-connected data points will give algorithms information about what will not provide additional insight. After the clustering phase, we will have new data sets that we can pass along to our prediction models.

4.3 Machine Learning Predictions

The main way to increase safety in a VANET is to use data analysis to create an actionable item. Although clustering can relate our data together, it does not provide a mechanism for decision making. Machine learning regression models, or predictive models, will allow us to predict possible vehicle locations in order to gain actionable insight on VANET data. The predictions will mitigate risk by validating locations of vehicles. We can use this information to make decisions about possible hazards for vehicles in the network. The multi-layer approach of clustering before making predictions will increase the accuracy and decrease error for our machine learning regression models.

4.4 Concept of Operations

The ultimate goal of the multi-layer system is make roadways safer. To achieve high level of safety, a user can specify an are that they would like to cover. For our VANET case, the user is likely a government entity and the area covered would be a city (i.e. City of Chattanooga tracking vehicles within city limits). A single centralized hub, call it a vehicle operations center, would have a map view of the entire desired area and all the vehicles in that given network. Clusters would be identified and color-coded. Vehicle predictions could be displayed in both time based interfaces and map interfaces. Possible attacks would be identifiable and verifiable through the interfaces, and the user can take corrective action.

CHAPTER 5

Simulation Setup and Implementation

In this chapter, we explain how we developed our simulation and clustering algorithm parameters. Section 5.2 explains the simulation we used as well as the setup. We discuss our predictive models in Section 5.2 as well as an overview of how our system is pieced together. Each of our clustering implementations are discussed in Section 5.3.

5.1 Simulation Setup

5.1.1 OpenStreetMap

Simulation of Urban Mobility allows for many different types of possible inputs to generate a simulation. OpenStreetMap, used in our simulation, is an open-source map data generating software. OpenStreetMap provides data about streets, highways, intersections, trails, railways, parking, and much more [38]. OpenStreetMap creates a *.OSM* file that contains all street data to be passed to our SUMO simulation. SUMO can then take the map and environment data and create real-world driving simulations on top of the map.

5.1.2 Simulation of Urban MObility

Simulation of Urban MObility is an open-source platform supported by the Eclipse Public License and the GNU General Public License [15]. SUMO provides us with a mecha-

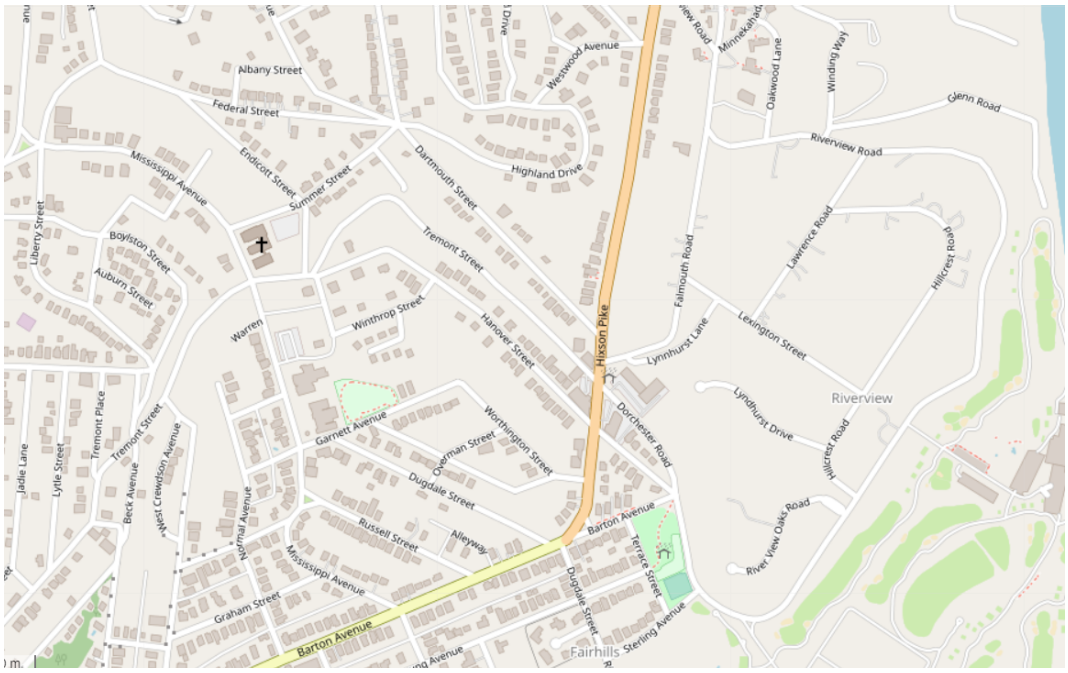


Figure 5.1 OpenStreetMap of North Chattanooga used in our simulations

nism to accurately simulate cars moving through a city. Real-world connected vehicle data does not yet fully exist in a form suitable for our work. SUMO Simulations can be created in a number of different ways, but as previously stated, our simulation will be created using an OpenStreetMap file. Using the command line and a *netconvert* command, the *.OSM* file is converted to an *.XML* network file that SUMO can process. From there, SUMO has quite a few built in tools that make networks into usable simulations - one of which is a Python script *randomTrips.py*. *randomTrips.py* generates random trips for vehicles initialized in the configuration file [15].

A SUMO Configuration file requires a few parameters to defined in order to run: a network file, a route file, duration, number of vehicles, and output file type. In our case, we used Floating Car Data (FCD) as our output file type. SUMO supports many different output types including emissions data, railway data, and large raw data files - Floating Car Data was chosen because the output focuses specifically on vehicle position data [15]. This gave us a roughly 1MB .CSV file with 12,383 rows. This may seem like a relatively

Figure 5.2 The map of our portion of Chattanooga in SUMO. Black lines represent main road while grey are smaller side roads

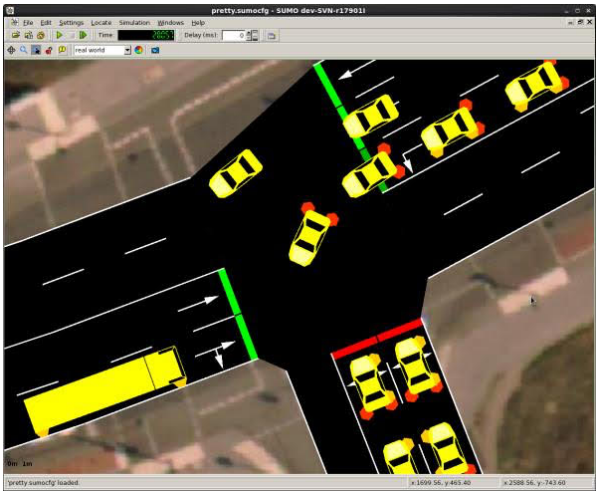


Figure 5.3 A sample intersection in SUMO. Vehicles are yellow, and traffic light colors are shown at the end of a lane

small data-set. However, BIRCH hierarchical clustering, although designed for larger data sets, is relatively memory costly (requiring n^2 memory) and can use all available memory if possible due the hierarchical nature of the algorithm [20, 21]. Running our simulation will give us a large accurate data set that represents real-world traffic patterns. A snapshot of our pre-clustering data set can be seen in Table 5.1 where the variables define by SUMO

documentation are:

- timestep_time - The current time of the simulation in seconds.
- vehicle_angle - The relative angle or direction of the vehicle to 12:00 position
- vehicle_id - The ID number of the vehicle
- vehicle_pos - The position of the vehicle relative to the start of the current lane
- vehicle_speed - Current speed of the vehicle in meters per second
- vehicle_x - The current X value of the vehicle relative to the entire map
- vehicle_y - The current Y value of the vehicle relative to the entire map

Table 5.1 Initial Data Set from SUMO. Later, our clustering algorithm will add an addition feature called Cluster

timestep_time	vehicle_angle	vehicle_id	vehicle_pos	vehicle_speed	vehicle_x	vehicle_y
0.00	9.83	0	5.10	0.00	558.77	1336.89
1.00	9.83	0	6.61	1.51	559.03	1338.38
1.00	292.35	1	5.10	0.00	2112.84	2170.77
2.00	9.83	0	10.31	3.70	559.66	1342.02
2.00	292.35	1	6.85	1.75	2111.22	2171.44

5.2 Predictive Models

Sci-Kit Learn has many models that can be used to predict target variables. However, much fewer of them support multiple outputs [2]. Our case is to predict vehicle location based on an X and Y value pair relative to our map. The application of these can be seen in Figure 5.2. To do this, we need machine learning algorithms that can support multiple output predictions. The models examined in this thesis are as follows:

- *LinearRegression()*- LinearRegression is a relatively simple algorithm that tries to create two "best-fit" lines for our target variables [25]. The trend line created becomes a great

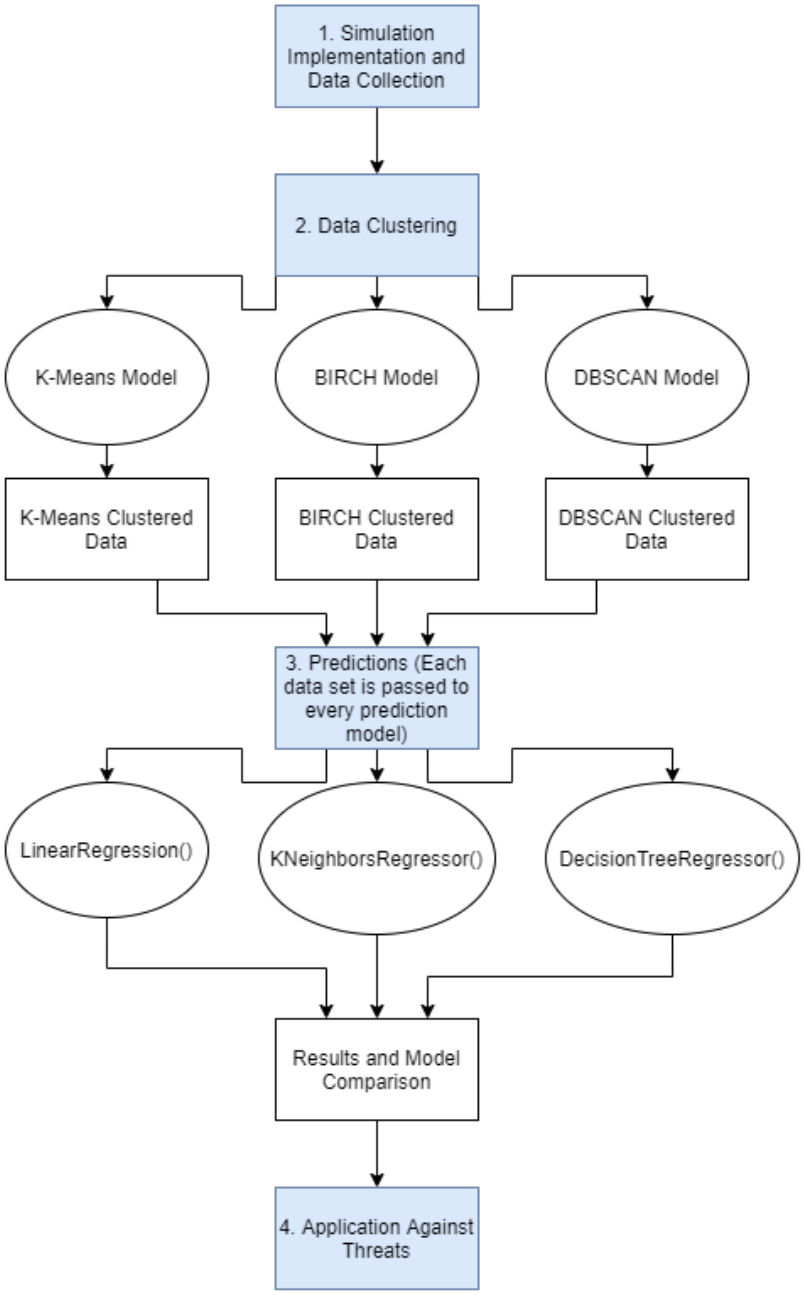


Figure 5.4 The steps used to create our model. Each blue box represents a major step. The machine learning models are represented by circles

baseline tool to compare other models to which is the key reason we chose to implement LinearRegression() [25].

- *KNeighborsRegressor()* - Using a Euclidean distance calculation similar to K-Means

between data points, KNeighborsRegressor predicts future points by distance similarity to current points [2]. KNeighbors was chosen because of it's strong performance with specifically numeric values, and X and Y in our case with be numeric coordinates. [2].

- *DecisionTreeRegressor()* - DecisionTreeRegressor creates a tree structure of all data points and makes predictions based on previous data's "Decisions" [2]. DecisionTree was chosen because it is one of the few supervised learning methods that does not require additional data preparation.

We will use each of these models to created a predicted X and Y value which we can examine for accuracy. In addition, we can apply our predictions to our threat model to isolate any anomalies or malicious behavior.

5.2.1 Pre-clustering Baseline

To establish a baseline and prove that clustering improves our predictions, we will show our prediction models without the extra *Cluster* variable first. A sample of this data set can be seen in Table 5.2 where the variables to be predicted are *vehicle_x* and *vehicle_y*.

Table 5.2 Non-clustered data set sample with isolated X and Y values

timestep_time	vehicle_angle	vehicle_id	vehicle_pos	vehicle_speed
0.00	9.83	0	5.10	0.00
1.00	9.83	0	6.61	1.51
1.00	292.35	1	5.10	0.00
2.00	9.83	0	10.31	3.70
2.00	292.35	1	6.85	1.75

We run each of our predictive models on the full version of the data set. Our results are shown in Table 6.1. Percentage Accuracy is calculated by dividing the Mean Absolute Error into the average X and Y value [39]. We will use these numbers as a baseline to compare to our other stacked models.

Table 5.3 Prediction Results without a Cluster variable. We can see that our error relative to actuals as well as accuracy

Predictive Model	Mean Absolute Error	Percentage Accuracy
LinearRegression() - No Cluster	300.449	88.50%
KNeighborsRegressor() - No Cluster	167.575	93.59%
DecisionTreeRegressor() No Cluster	20.029	99.24%

5.3 Clustering Implementations

Each clustering algorithm has it's own unique set of parameters that must be calculated and implemented correctly before clustering can happen. The parameters ultimately determine a total number of clusters that our data set will be divided into - it is extremely important to note that each algorithm will behave differently for any given unique data set. Thus, there is no exact formula to determine the optimal number of clusters [2]. This section shows our setup and reasoning for each clustering algorithm parameters to ensure accuracy of the implementation.

5.3.1 K-Means Implementation

K-Means provides one of the quickest and easiest implementations for any clustering algorithm [18]. In addition, K-Means is agnostic to the size of a given data set - it can perform equally accurate and quick calculations for small and large data sets [18]. It calculates the Euclidean distance to each neighbor and clusters them [2, 19]. We chose K-Means because of it's simplicity, ability to scale with data sets, and it's use as a baseline clustering algorithm [18]. Any K-Means clustering algorithm will require a number of clusters desired as an input parameter [2]. There is no guaranteed optimal number of clusters, and obviously, each data-set passed into a K-Means clustering algorithm will have it's own unique output [2]. However, the Elbow Method is the most prominent way of determining a best fit number of clusters for a data-set [40]. The Elbow Method for our data-set is shown below.

The Elbow Method calculates *Distortions* for each possible number of clusters. *Dis-*

Figure 5.5 k represents the number of clusters for a given Distortion value D. The point of maximum curvature is 3 which will be our optimal k

tortions are defined as the average of squared Euclidean distances to the center [40]. The point of most curvature is known as the Elbow and provides insight to the likely optimal number of clusters [40]. The "Elbow" acts as somewhat of a point of diminishing returns, so in our case, looking at Figure 5.5 moving past more than 3 clusters will provide us no more of an accurate model [40]. Thus, our optimal number of clusters is 3. Passing our 3 clustering argument to our data-set and plotting data points of our X and Y vehicle values, we get the clusters shown in Figure 5.6.

We can see that there is a pretty uniform divide of each cluster which is pretty common in K-Means visualizations [18]. By clustering with K-Means, we have created an additional grouping variable, *Cluster*, based on all of the available data for all vehicles from SUMO. We can now pass our new data set to our prediction models.

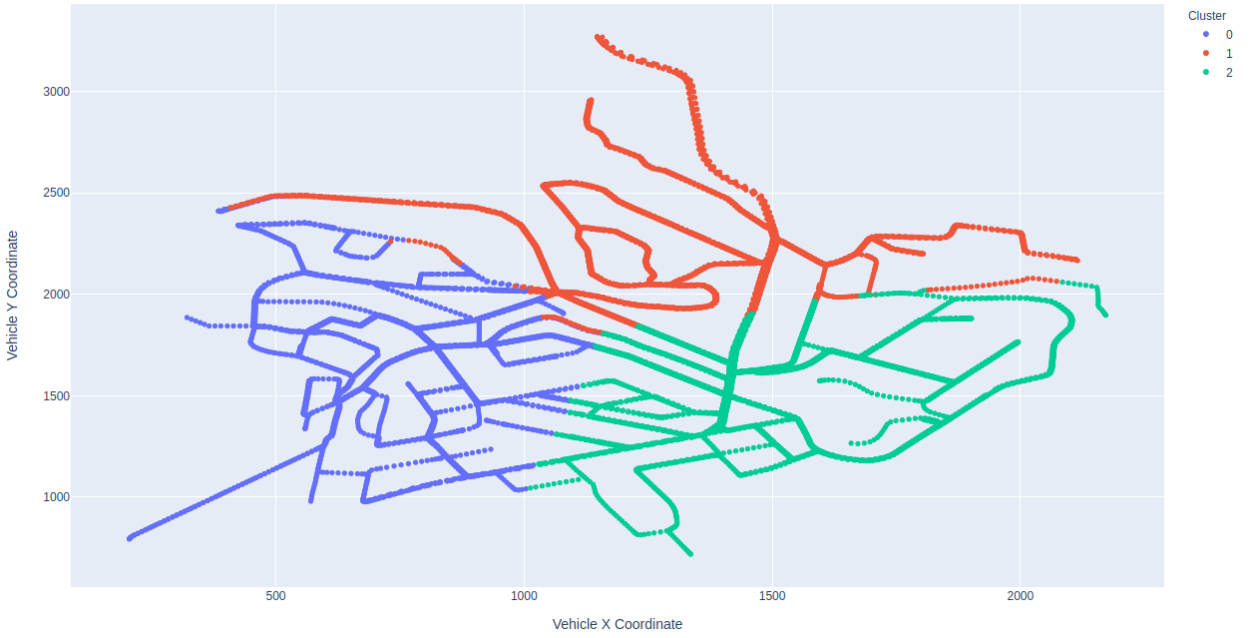


Figure 5.6 Our K-Means Clustered Data Set. Each color represents a different cluster

5.3.2 BIRCH Clustering Implementation

BIRCH Clustering creates hierarchies of each data point based on similarity [21]. Creating hierarchies means that data points will be grouped all as one (top of the hierarchy) and broken down into clusters moving down the hierarchy until each data point is it's own cluster [21]. BIRCH was chosen for two main reasons: it's success with large data sets [20] and it's ability to begin clustering and learning without a full data set [21]. VANETs have a massive amount of data moving through them and vehicles may not always be able to propagate full information through the network, so BIRCH fits well with our use case. As mentioned, hierarchical clustering algorithms create a hierarchy of all the data points - from there, a set number of clusters can be decided on from a dendrogram [22]. Dendrograms are a visual representation of the clusters correspondence as they move from less to more clusters (or vice versa) [22]. Take for example Figure 5.7.

You can see that there are clusters each major region of the United States (with some

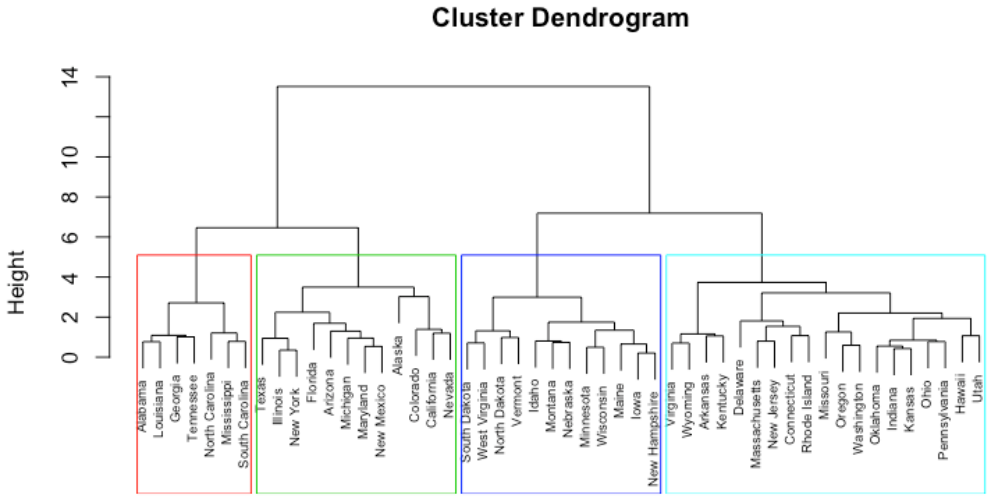


Figure 5.7 Dendrogram Cluster of 50 States

outliers), but for the most part it is clustered into Southeast, Northeast, Mid-west, and West. Furthermore, moving down the dendrogram, you could cluster states as pairs like Tennessee and Georgia. Or, moving up, you could cluster as just East Coast and West Coast. The point is that dendrograms give you several different options to cluster data. In our case, data set is much larger than 50 data points, so the dendrogram can become slightly harder to interpret. The dendrogram for our data set is shown in Figure 5.8

Figure 5.8 shows all of our data points be clustered from one single cluster down to each having its own cluster, all 12,383 rows are a single cluster. Our case presents a more challenging environment to determine an optimal number of clusters. Zooming in, we were able to determine about 3 possible clusters at *height=19*, another 5 at *height=17*, and roughly 7 more at *height=15* for a total optimal cluster number of 15 to align with the guidelines set in [22]. It is a coincidence that our height and number of clusters happened to be equal at 15. As a note, this is an educated guess as it became difficult to zoom in with so many data points on a relatively small graph. However, this method aligns with the assumption that there is no exact formula for determining an optimal number of clusters [2].

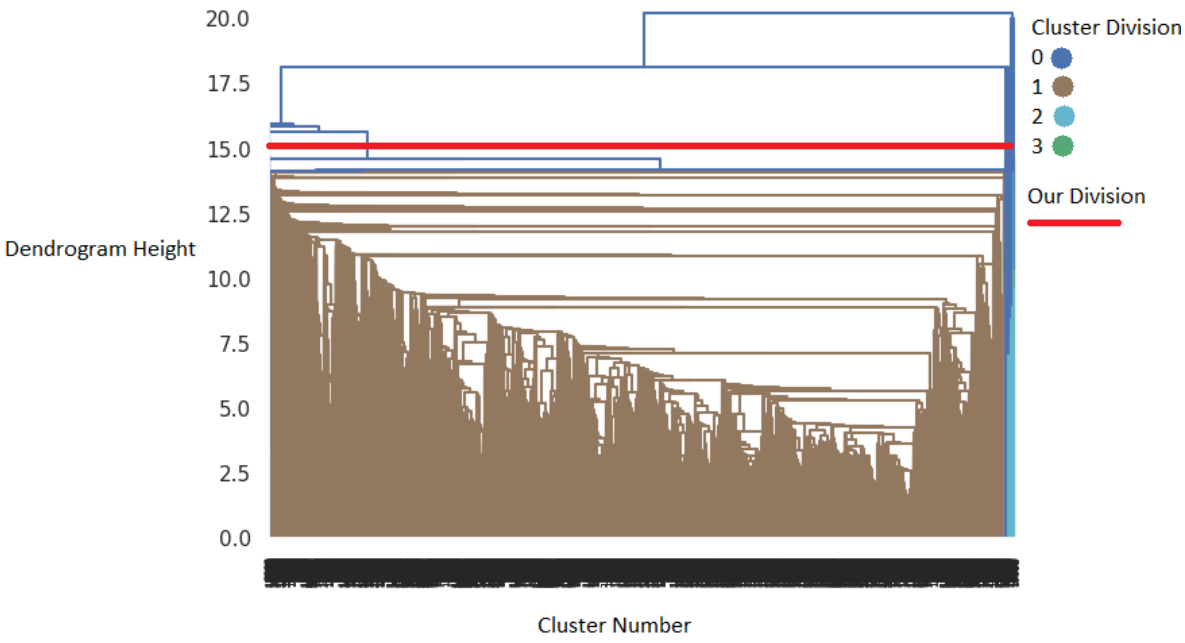


Figure 5.8 Dendrogram of our data set. A single cluster for all data points is represented at the top, and clusters are divided moving down the hierarchy until each data point is its own cluster

We can now take our 15 cluster idea and run the algorithm on our data set. Plotting our vehicle position and the clusters for that X,Y value pair, we get Figure 5.9. We can see that relative to K-Means there are more clusters that appear more randomly distributed.

5.3.3 DBSCAN Clustering Implementation

DBSCAN essentially takes a calculated density and applies it to a data set to create clusters [2, 23]. It is one of the few supervised learning algorithms that requires almost no data preparation [2]. DBSCAN was chosen due to it's success with anomaly detection, minimal data preparation, and low computational time and memory cost [23]. In real-world vehicular networks, a given vehicle cannot waste time preparing data and will be limited computationally relative to non-mobile computers. DBSCAN does not require a number of clusters in order to implement the algorithm - the algorithm itself creates the clusters [2].

Figure 5.9 BIRCH Clustering of our map. Each different cluster is shown as a different color

The two crucial parameters for implementing DBSCAN clustering are determining the proper *Epsilon* and *minimum_samples* values [2, 23]. The *Epsilon* value is the distance value where two points will be considered neighbors - similar to distance classifications we saw in K-Means and KNeighbors [2]. The other parameter required for DBSCAN is *minimum_samples*. *minimum_samples* is the number of neighbors that a data point can have to be part of a cluster. *Epsilon* and *minimum_samples* combine together classify the density of data points required to create clusters [2]. To determine our *Epsilon* value, we can, again, use the Elbow Method and plot the Euclidean distances for each point [23].From Figure 5.10, we see that our point of of maximum curvature is at about 30, so we used an *Epsilon = 30* for our clustering.

minimum_samples on the other hand, is more of a tolerance-like value dependent on the data set and how you want to define "density". And again, in clustering, there is no exact way to determine clusters as it is dependent on the data set [2]. The default for DBSCAN is 5, but through some testing and plotting of different values, we determined

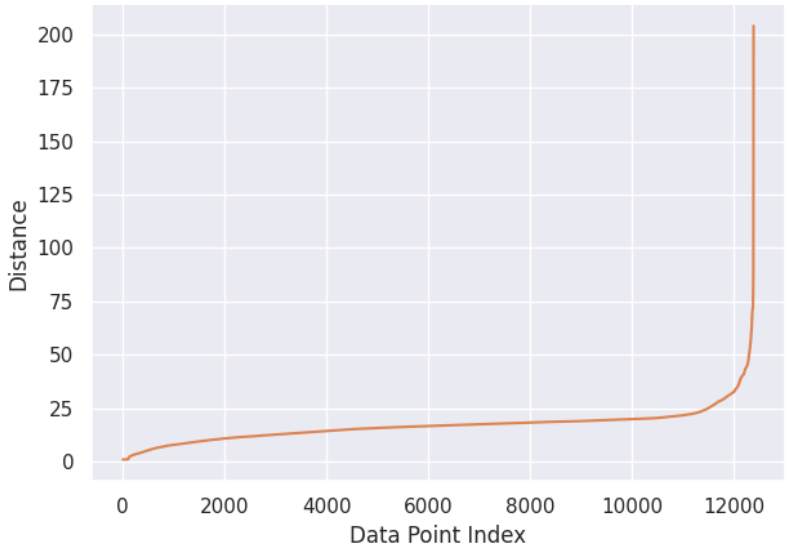


Figure 5.10 Elbow Method for DBSCAN. The point of maximum curvature will show us the optimal number of clusters, so we chose 30 for Epsilon value

minimum_samples = 7 was the best fit for my data set. 7 allowed a sufficient number of clusters to be created without classifying to many data points as outliers. We can see our clustered map in Figure 5.11

Since DBSCAN focuses on density to create clusters, it is extremely important to point out that in Figure 5.11 each light purple point is actually determined to be an outlier by the algorithm and not included in any cluster. These points did not meet the density requirement. We can plot the vehicles and their clusters over time to get a better understanding of how DBSCAN is working.

Figure 5.12 plots the vehicles clusters as they move through time in the simulation. The first point is that DBSCAN creates many more clusters than any algorithm implemented thus far. This is due to the fact the clusters are only created when the density requirement is met. So, when vehicles are much closer together in something like an intersection, a new cluster will be formed. When the vehicles continue on their path and the density is broken, that cluster will no longer exist. The vehicles will move back to *Cluster = -1*, or outlier, until they are in a situation where the density requirement is met again and a new cluster is

Figure 5.11 DBSCAN Clustered Map. The purple data points are non-clustered vehicles, and clusters appear more common at intersections. We expect more clusters to appear in more dense areas

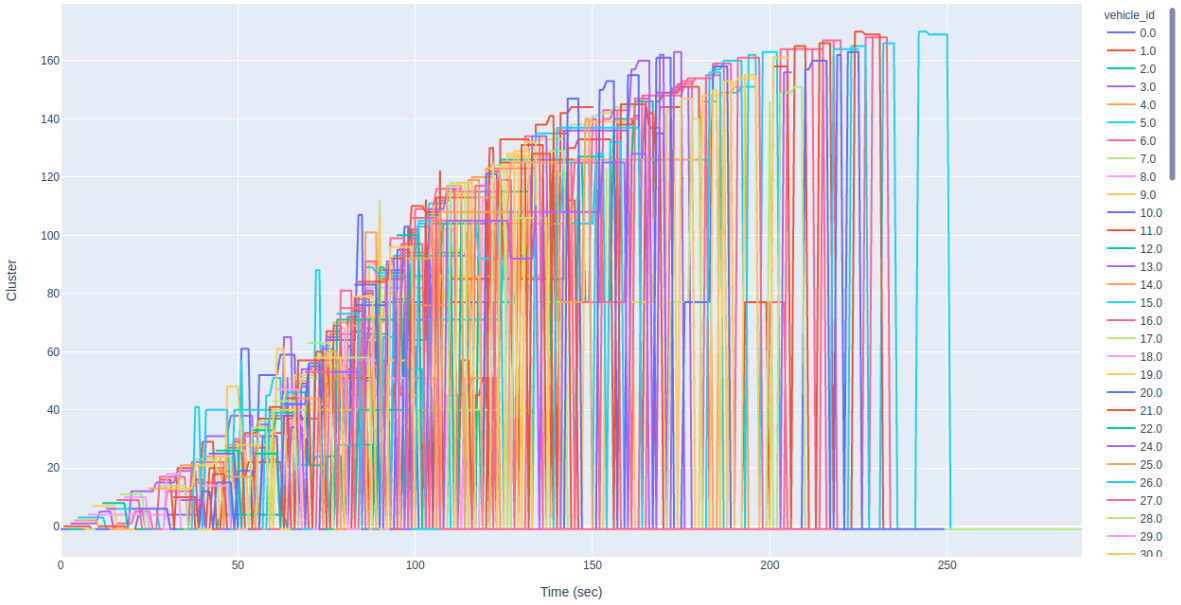


Figure 5.12 DBSCAN Vehicle Clusters. Vehicles create new clusters when they enter more dense areas. This shows vehicles moving out of *Cluster=-1*, or outlier, to clusters

formed. You can see the density in action in Figure 5.11 - intersections very clearly contain most of the clusters. A further examination of the phenomena will occur in Chapter 6.

We are now prepared to properly pass our clustered data sets to prediction models.

CHAPTER 6

System Evaluation

This chapter implements our prediction models and compares results for each clustered data set. Section 6.1 reviews our non-clustered results. Our K-Means results are in Section 6.2. BIRCH results are discussed in Section 6.3, and DBSCAN results are presented in Section 6.4. Lastly, we look at our threat model in 6.5.

6.1 Review of Non-Clustered Baseline Predictions

The goal of this paper is to find a multi-layer clustering and prediction model that enhances our ability to identify vehicle locations through clustering. But, we also need to show that these models out perform a non-layered approach - in other words, making predictions without clustering. We performed and showed baseline results in Chapter 5, but it is worth showing the summary numbers again at the beginning of this chapter as they are referenced quite a bit.

Table 6.1 Prediction Results without a Cluster variable

Predictive Model	Mean Absolute Error	Percentage Accuracy
LinearRegression() - No Cluster	300.449	88.50%
KNeighborsRegressor() - No Cluster	167.575	93.59%
DecisionTreeRegressor() No Cluster	20.029	99.24%

6.2 K-Means Predictions

To make predictions on data, you must isolate your target variables from your predictor variables [2]. Target variables in our case are our vehicle X and vehicle Y values, and our predictor variables are vehicle angle, ID, position, speed, time, and cluster. After isolating our vehicle X and vehicle Y to a test array, our data set before predictions is shown in Table 6.2. A summary of our results with Mean Absolute Error and Predictive Accuracy can be seen in Table 6.6.

Table 6.2 K-Means Clustered Data Set with our new *Cluster* variable correlating all data points

timestep_time	vehicle_angle	vehicle_id	vehicle_pos	vehicle_speed	Cluster
0.00	9.83	0	5.10	0.00	1
1.00	9.83	0	6.61	1.51	1
1.00	292.35	1	5.10	0.00	2
2.00	9.83	0	10.31	3.70	1
2.00	292.35	1	6.85	1.75	2

6.2.1 K-Means with LinearRegression()

Now, running our prediction models we are able to create a predicted X and predicted Y value for each of the models passed to our clustered data set. First, we examine the LinearRegression() model. The predicted data set is shown in Table 6.4 and Table 6.5.

Table 6.3 Sample of K-Means Clustering LinearRegression Predictions

timestep_time	Cluster	Linear X Predicted	Linear Y Predicted
0	2	949.7564186	1562.178796
1	2	952.7636795	1558.130525
1	0	1467.402229	2079.564377
2	2	956.0943624	1553.06635
2	0	1470.7926	2074.854324

Now, we can plot these predictions to see how models performed over the entire sim-

ulation. For ease of visualization purposes, we are only going to plot the actuals vs predicted values for Vehicle 0 over time (Figure 6.1 - if we plot all 100 vehicles, time graphs become overloaded with information and difficult to interpret. The map does however represent every vehicle in the simulation. With that, we will plot these as an X, Y overlay of our actual X and Y map. The plots for actual values vesus predicted values can be seen in Figure 6.1 and Figure 6.2.

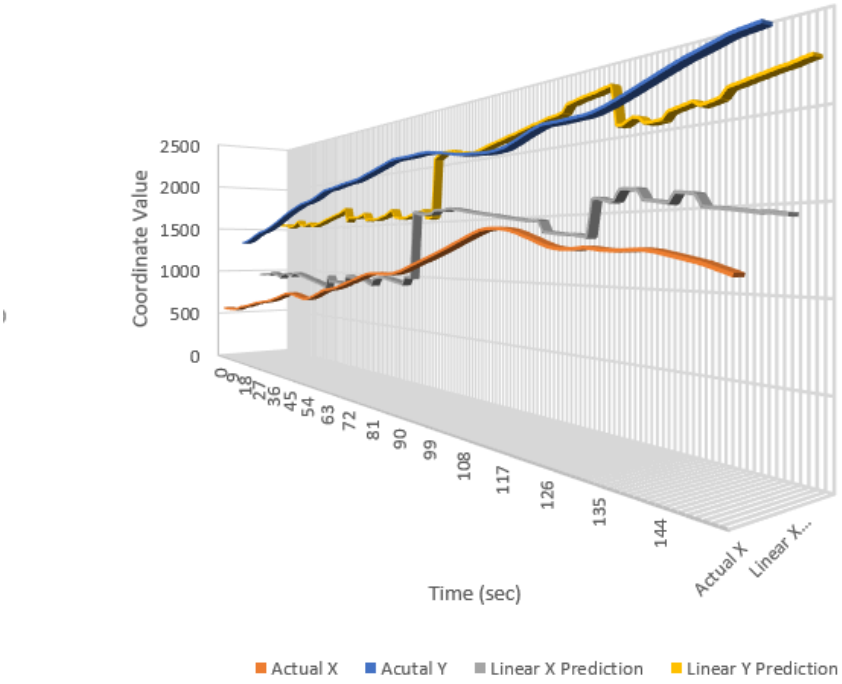


Figure 6.1 K-Means Actual vs Predicted for LinearRegression. This graph only represents vehicle 0 over time, and we can see the trendline that LinearRegression makes

These plots accurately reflect what we see in the results summary. *LinearRegression()* predicted values are relatively inaccurate - meaning they do not fall directly on the actual plots as frequently. In addition, when they are wrong (error), it is quite far off from the actual. In Figure 6.1, we that the prediction is somewhat of straight line across the data (with a few jumps for correction). This accurately represents our notion that *LinearRegression()* predicts by creating a single trend line through the data. This idea is further demonstrated

Figure 6.2 K-Means Actual vs Predicted for LinearRegression. Every vehicle is represented in this figure, and the trend line centers all of our predicted data

in Figure 6.2. All the predicted data points are thought to be in the direct center of the actuals - or, somewhat of an average of every data point. The trend line creates our error and accounts for the inaccuracy because it tries to do a basic modeling of every data point.

6.2.2 K-Means KNeighbors Predictions

Taking the same, K-Means Clustered data set, we can run our *KNeighborsRegressor()* over the data set. A sample of the prediction values can be seen in Table 6.4.

Table 6.4 Sample of K-Means Clustering KNeighbors Predictions

timestep_time	Cluster	K-Nearest X Predicted	K-Nearest Y Predicted
0	2	903.52	1442.228
1	2	903.52	1442.228
1	0	2109.354	2172.206
2	2	731.262	1392.694
2	0	2109.354	2172.206

We can again plot our actual and predicted values both overtime and relative to our

map. Our K-Means Clustered plots can be seen in Figure 6.3 and Figure 6.4. K-Means over time seems to relatively accurately follow the actuals over time seen in Figure 6.3. However, we see less severe case of centering on the maps for *KNeighbors*. But, the data points still appear erratic and random around the middle. This reflects the nature of the algorithm. Since points in the middle are closer together, the Euclidean distance calculated by *KNeighbors* is smaller in that area with many possible options (meaning there are known data points above, below, and on each side). So, it predicts every data point to be relatively near the other creating the error. Looking further out of Figure 6.4, the Euclidean distances are greater and the possible options are fewer. Thus, it is able to predict very accurately of a few paths heading out of the simulation.

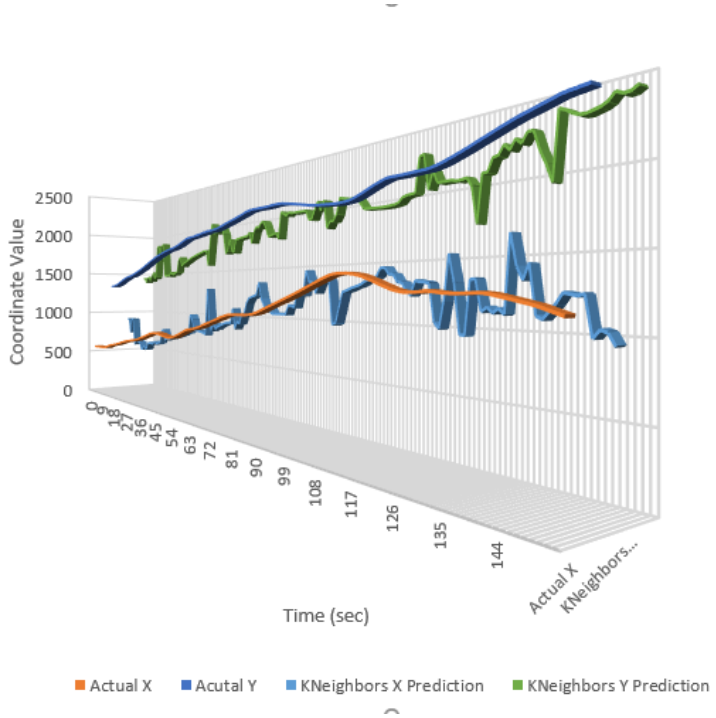


Figure 6.3 K-Means Actual vs Predicted for K-Nearest Neighbors. Predictions center around the actual, but do have some error

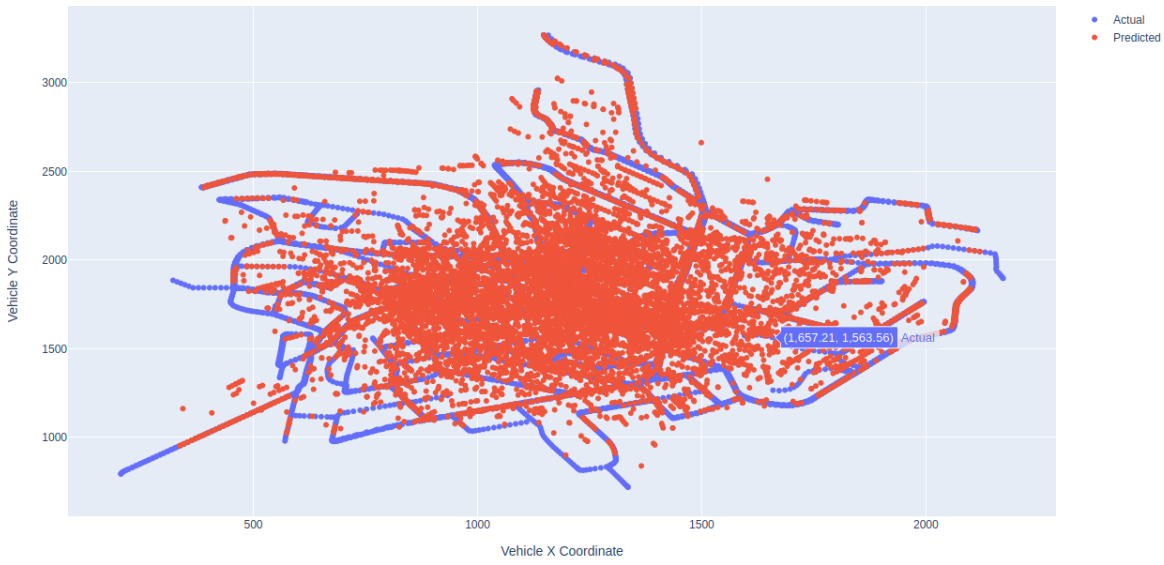


Figure 6.4 K-Means Actual vs Predicted for K-Nearest Neighbors. Predictions are worse in the center because of small Euclidean distances to near-by points

6.2.3 K-Means DecisionTree Predictions

The last regression model we will examine on our K-Means data set is the *DecisionTreeRegressor()*. A prediction sample is shown in Table 6.5.

Table 6.5 Decision Tree Results. The errors appear to be low relative to our originals

timestep_time	Cluster	DTree X Predicted	DTree Y Predicted
0	2	558.77	1336.89
1	2	559.03	1338.38
1	0	2112.84	2170.77
2	2	559.66	1342.02
2	0	2111.22	2171.44

We will create the same time and map plots we did before to examine our data. Looking at Figure 6.5 and Figure 6.6, we see that *DecisionTreeRegressor* much more accurately predicted our vehicle locations. We are dealing with a very low error relative to our X and Y coordinate values (26 vs 2600), so we only see small deviations from the actual. Because of the low error, Figure 6.6 was plotted as a 3D line graph to show actuals next to predicted values and better visualize errors. The DecisionTree plot only shows a few points that are

slightly off. The map plot in Figure 6.6 is virtually impossible to see any mistakes due to low error relative to the size of the map. Based on our plots and summary, *DecisionTree* appears as the best predictive algorithm for our K-Means Clustered data set.

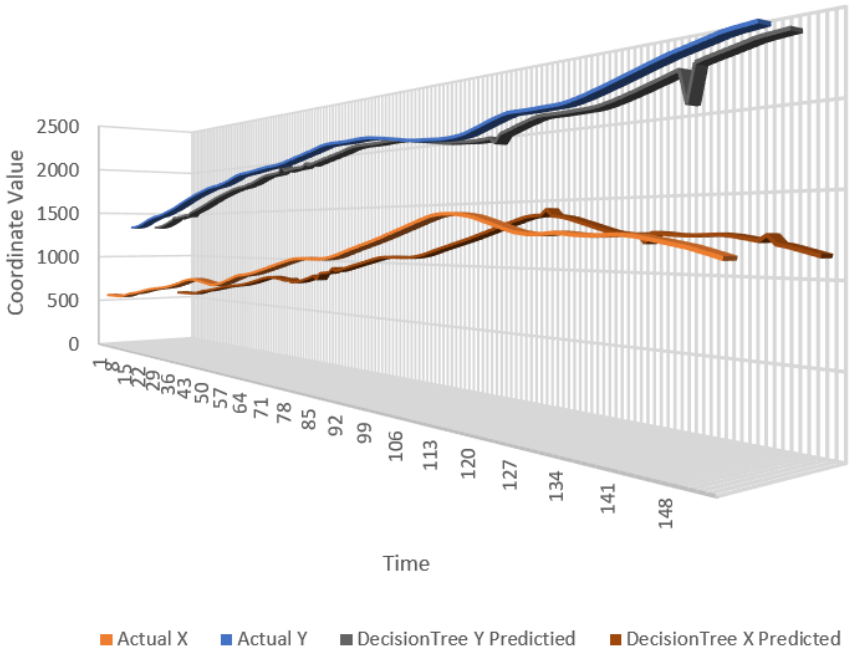
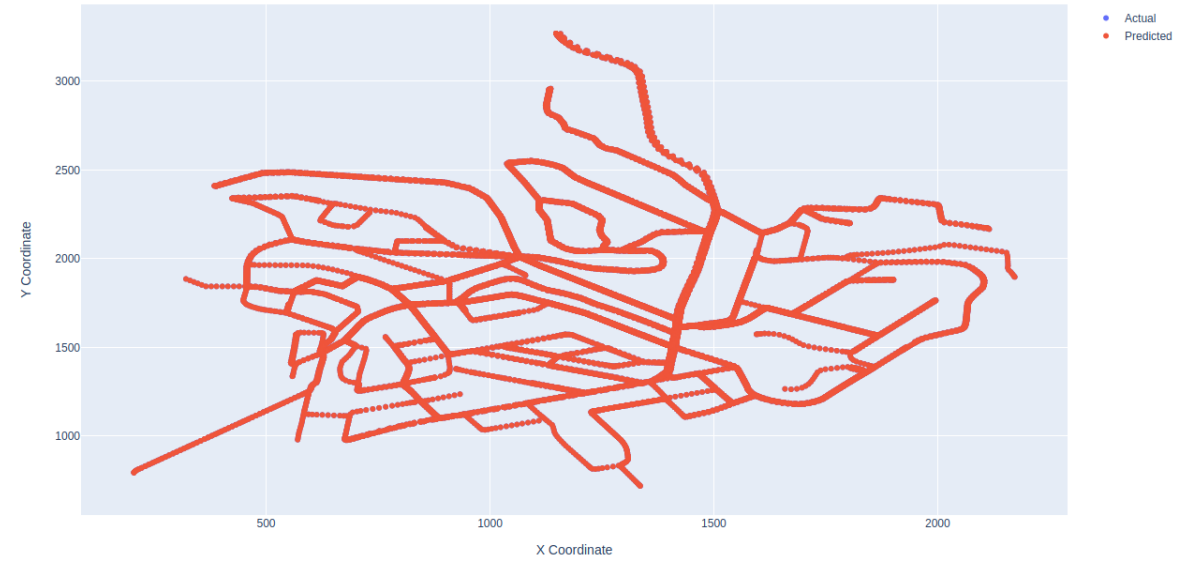


Figure 6.5 K-Means Actual vs Predicted for Decision Tree. Predicted points fall very close the actuals

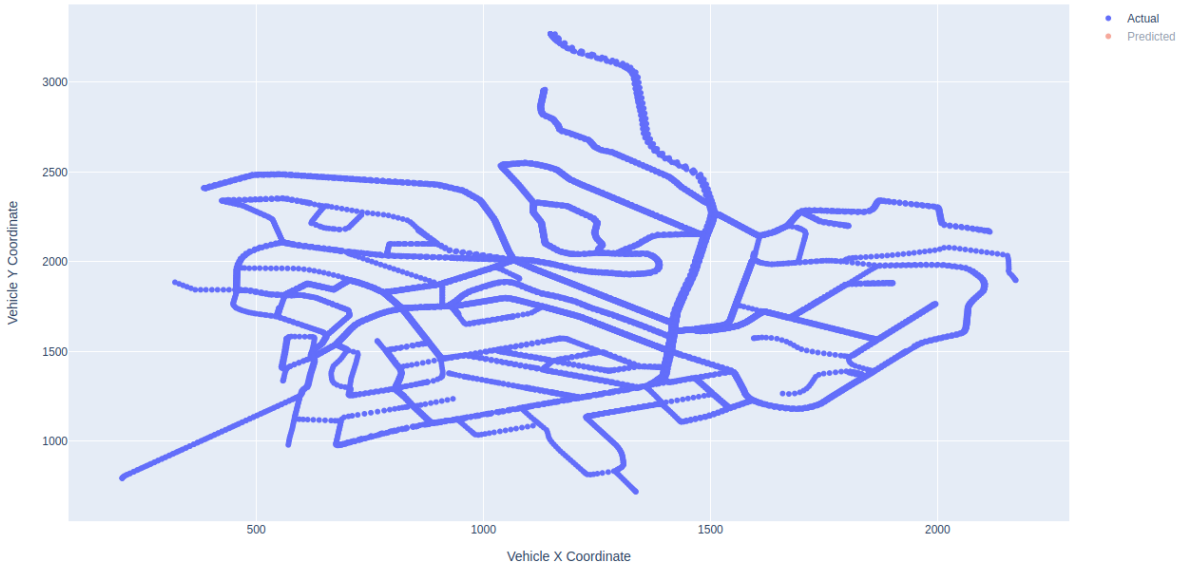
Table 6.6 Summary of K-Means Prediction Results with *DecisionTreeRegrssor* showing the lowest error and highest accuracy

Predictive Model	Mean Absolute Error	Percentage Accuracy
K-Means LinearRegression()	249.53	90.45%
K-Means KNeighborsRegressor()	167.08	93.60%
K-Means DecisionTreeRegressor()	26.671	98.98%

From Table 6.6, we can see that, relative to our non-clustered baseline, *LinearRegression()* performs quite a bit better while the other two algorithms match or fall short of the non-clustered predictions. This is caused by the relatively few number of clusters recommended by our Elbow Analysis. The few number of clusters averages *LinearRegression* closer to the true value [25,41]. However, *KNeighbors* and *DecisionTree* do not have enough



(a) Predicted Values

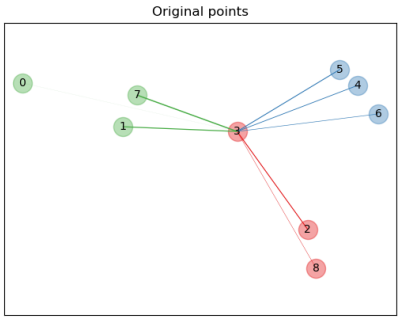


(b) Actual Values

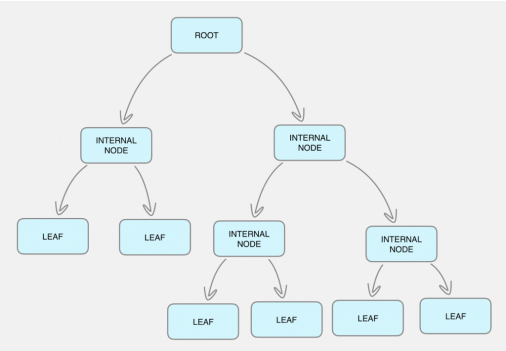
Figure 6.6 K-Means Actual vs Predicted for Decision Tree. We have almost a direct overlay of the map. Small deviations in predictions are hard to see. It is important to note that inaccurate predictions could fall in an area where another point is located as well

possible options of clusters to learn on and improve predictions [41]. We can see in Figure 6.7 how this works out. *KNeighbors* has 3 possible points (our case as well) to branch to to

create predictions from. If provided with more possible options, it could potential return a more accurate prediction. In the case of *DecisionTree*, internal nodes are created every time a decision is made. Decision Trees a able to more accurately model data with more decisions to learn on [2, 41]. We will observe the other side of this phenomena in cluster algorithms with more clusters, but at this point, we have not been able to create a more accurate model using K-Means clustering.



(a) K-Neighbors



(b) Decision Tree

Figure 6.7 Decision Making for KNeighbors and Decision Tree Algorithms. With limited options, prediction algorithms can only be helped so much by K-Means clustering

6.3 BIRCH Clustering Predictions

Taking our BIRCH implementation from Chapter 5, we can now run the BIRCH clustering algorithm over initial data set 5.1 to create an additional *Cluster* variable Like our K-Means, we now have a data set with a cluster variable created from all points in the data set. But, clusters are much different because of the requirements and implementation differences of the algorithms - a sample of our new data set is shown in Table 6.7. Again, we will isolate our target variables and pass our new data set through each predictive model in order to enhance our predictive ability.

Table 6.7 BIRCH Clustered Data Set showing more possible values for our *Cluster* variable

timestep_time	vehicle_angle	vehicle_id	vehicle_pos	vehicle_speed	vehicle_x	vehicle_y	Cluster
0.00	9.83	0	5.10	0.00	558.77	1336.89	11
1.00	9.83	0	6.61	1.51	559.03	1338.38	11
1.00	292.35	1	5.10	0.00	2112.84	2170.77	0
2.00	9.83	0	10.31	3.70	559.66	1342.02	11
2.00	292.35	2	6.85	1.75	2111.22	2171.44	3

We can now plot these results to visually compare. Like K-Means, for visualization purposes, we are only plotting for Vehicle 0 on the time-based graphs. We can see an accurate representation of what we predict for the entire data set summarized in Table 6.11.

6.3.1 BIRCH LinearRegression Predictions

Our LinearRegression, although somewhat accurate in many cases like the Y prediction from *time=20* to *time=40*, has huge errors for the data points it inaccurately predicts - a good example is the Linear X prediction at *time=20*.

Our results show a decrease in accuracy for our *LinearRegression()* model, and at this point, I think it is safe to say that a linear regression model will not be the best solution for our data set due to the simplicity of the algorithm combined with the complex nature of our data set. The trend line design of *LinearRegression()* cannot account for all the complex changes from data point to data point. It has yet to perform at the level of the other algorithms in any case. Looking at each of the graphs for BIRCH LinearRegression, we again see too much centering of the predictions across the data.

Table 6.8 BIRCH Predictions. We see centering around *X=1125* and *Y=1600*

timestep_time	Cluster	Linear X Predicted	Linear Y Predicted
0	11	1110.631757	1518.287462
1	11	1119.113334	1523.445216
1	0	1144.976414	1949.147827
2	11	1131.008697	1532.609102
2	0	1154.70081	1955.14178

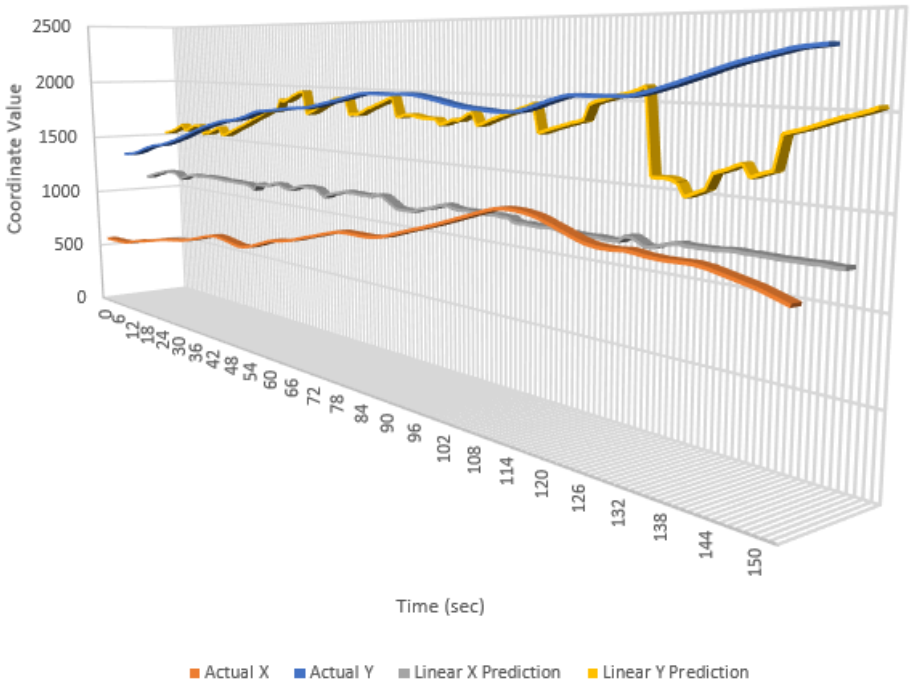


Figure 6.8 BIRCH Actual vs Predicted for LinearRegression. We see the LinearRegression trend line idea in action again

Figure 6.9 BIRCH Actual vs Predicted for LinearRegression. Our data points are centered because LinearRegression tries to create a "Best Fit" for all data points

6.3.2 BIRCH KNeighbors Predictions

For the *KNeighbors* plots, we see relatively little change from K-Means although the summary shows some improvement - we validate and justify the improvement later in this section. The time-based plot (Figure 6.10) shows quite a bit of accurate predictions, but there are pretty clear misses when the algorithm is unable to predict correctly. The map of actual vs predicted shows a similar case to K-Means the more central points are, like in the K-Means case, are less accurate than cases that occur more on the fringe of the simulation. It is becoming more and more obvious that the predictive algorithm used influences prediction outcomes much more so than clustering algorithms.

Table 6.9 BIRCH KNeighbors Predictions. Coordinate values more towards the edge of the map appear more accurate

timestep_time	Cluster	K-Nearest X Predicted	K-Nearest Y Predicted
0	11	903.52	1442.228
1	11	903.52	1442.228
1	0	2109.354	2172.206
2	11	731.262	1392.694
2	0	2109.354	2172.206

6.3.3 BIRCH DecisionTree Predictions

Lastly, the *DecisionTree* time-based plot (Figure 6.12) does show the small increase in accuracy. We see fewer and smaller deviations from the actual relative to the K-Means DecisionTree time-based plot. Mapping our actuals and predict values, we see a direct overlay although there are inaccuracies that are not directly visible in the graph but known from the summary.

Now, in the case of of *KNeighborsRegressor()* and our *DecisionTreeRegressor()*, we see improvement over K-Means by roughly a half a percentage point accuracy in each. More importantly, relative to our pre-clustering baseline, we see a half a percent improvement from

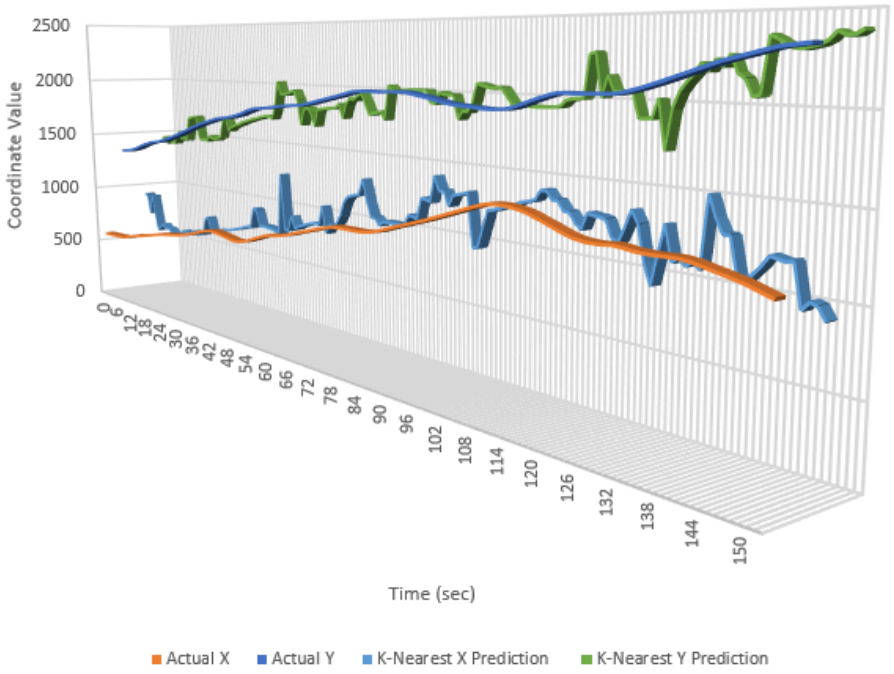


Figure 6.10 BIRCH Actual vs Predicted for KNeighbors. There are some small errors clearly visible

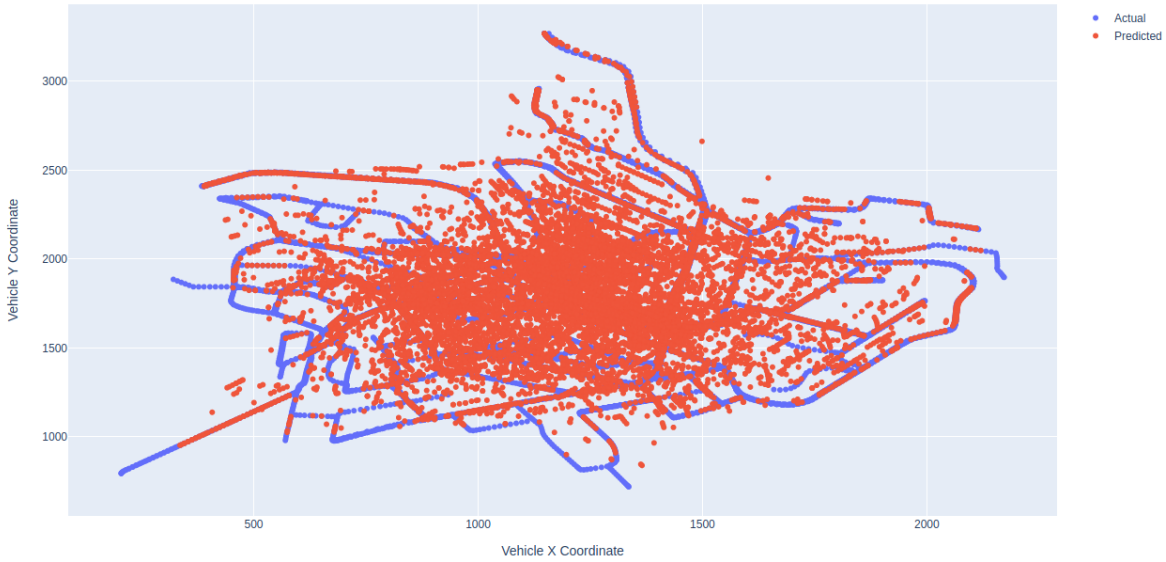


Figure 6.11 BIRCH Actual vs Predicted for KNeighbors. KNeighbors predicts data to be more central due to lower Euclidean distances in that area

Table 6.10 BIRCH Decision Tree Predicted Values

timestep_time	Cluster	D Tree X Predicted	D Tree Y Predicted
0	11	558.77	1336.89
1	11	559.03	1338.38
1	0	2112.84	2170.77
2	11	559.66	1342.02
2	0	2111.22	2171.44

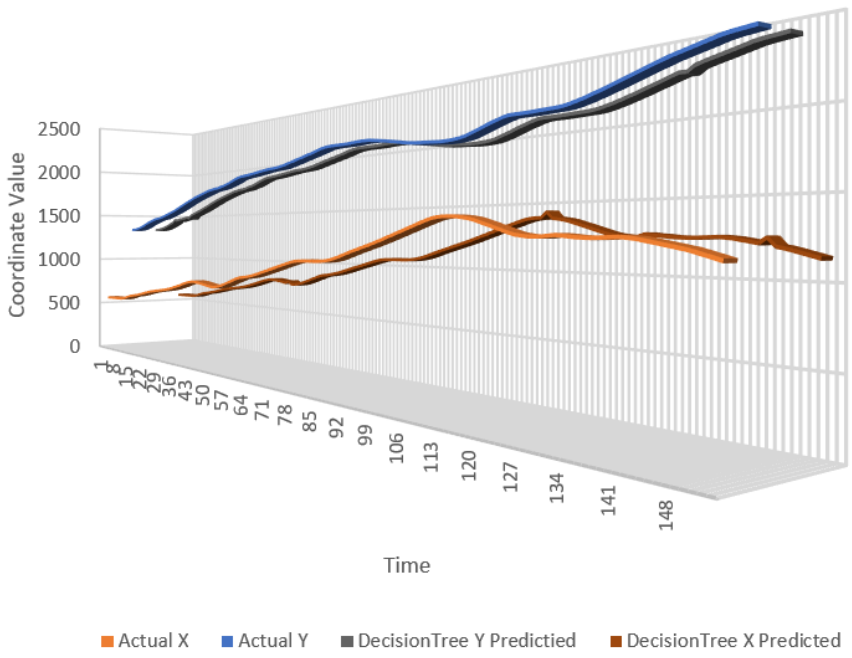


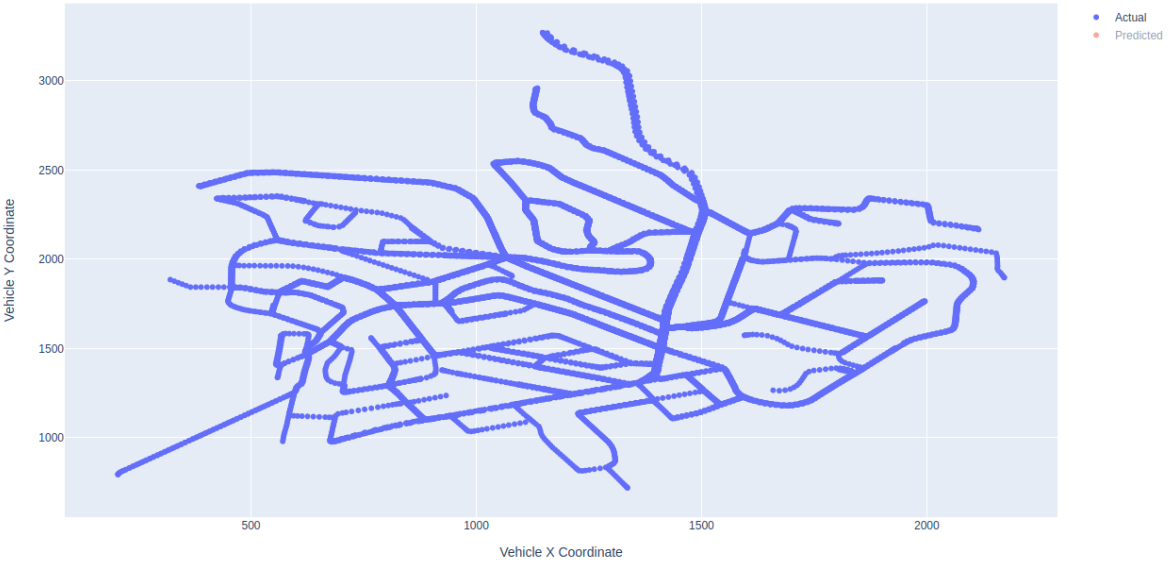
Figure 6.12 BIRCH Actual vs Predicted for Decision Tree. Only very small deviations from the actual occur

Table 6.11 BIRCH Clustering Results. Overall, our DecisionTree with BIRCH Cluster has scored the highest out of all algorithms thus far

Predictive Model	Mean Absolute Error	Percentage Accuracy
BIRCH LinearRegression()	283.97	89.20%
BIRCH KNeighborsRegressor()	155.68	94.04%
BIRCH DecisionTreeRegressor()	13.32	99.49%

KNeighborsRegressor() and about a quarter percent improvement from *DecisionTreeRegressor()*. Is this good? Half a percent, relatively, is not much in most cases. If you had half a percent of a take of gas left in your car, you would likely be close to being stuck on the side of the road. Modern machine learning cases generally are dealing with so much data

(a) Predicted Values



(b) Actual Values

Figure 6.13 BIRCH Actual vs Predicted for Decision Tree. We have almost a direct overlay of the map. Our predictions match our actuals very well

that small gains typically lead to big outcomes and best models can often be decided by just 0.01% [42, 43]. An real-world machine learning example of small percentage increase resulting in large occurred in the case of the Netflix Prize. Data teams competed to create the

best model to predict DVD recommendations - the winning team was awarded $1,000,000. Ultimately, the winning team beat out second place by 0.01% winning the million dollars, and third place was only a mere 0.15% off of second [43]. Similar machine learning cases do help show the significance of our 0.25% - 0.5% improvement, but it also can be further proven through simply thinking about our case in the real-world. A city, at any given moment, could have thousands of cars travelling through it's network. If we missing information about 0.5% of the total cars, we are losing location and speed information of potentially 100s of cars. Rouge cars in a VANET could be extremely dangerous to cars relying on information passed through the network - no location and speed information would result in collisions. Our case 0.5% example further proves the notion of small gains mean large impact in large data sets [42, 43]. BIRCH Clustering with a *DecisionTreeRegressor()* has improved our ability to predict and is currently the best multi-layer model for our case.

6.4 DBSCAN Clustering Predictions

Like our other algorithms, we can run DBSCAN clustering implementation derived in 5 over our initial data set to create a new set with a *Cluster* variable that represents groupings of all the variables. A sample of our DBSCAN data set with clusters is shown in Table 6.12. The key item to note in Table 6.12 is that Vehicle 0 is said to be in *Cluster = -1*. *-1* is DBSCAN's mechinism for classifying an outlier. So, at the beginning of the simulation, vehicle 0's data has not yet met the density requirements defined in the implementation to be classified in cluster. Vehicle 1, on the contrary, does start in a cluster even though it does not appear to be near Vehicle 0 initially - *Cluster = 0*. This is possible through the clustering of other variables like spped and position. Vehicle 1 could have been initiated driving slowly into a busy (or what will be busy) intersection. As we saw in Figure 5.12, vehicles will continuously move in and out of *Cluster = -1* as density based on vehicles around them changes. From here, we can run our predictive models, and a sample of results

similar to the other algorithms can be seen in Table and Table

Table 6.12 DBSCAN Clustered Data. We see a new cluster: *Cluster = -1*. This signals an outlier

timestep_time	vehicle_angle	vehicle_id	vehicle_pos	vehicle_speed	vehicle_x	vehicle_y	Cluster
0	9.83	0	5.1	0	558.77	1336.89	-1
1	9.83	0	6.61	1.51	559.03	1338.38	-1
1	292.35	1	5.1	0	2112.84	2170.77	0
2	9.83	0	10.31	3.7	559.66	1342.02	-1
2	292.35	1	6.85	1.75	2111.22	2171.44	0

6.4.1 DBSCAN Linear Results

We can plot actual values versus predicted values for each of our prediction tables to visually help validate what we see in our summary table for DBSCAN clustering (Table 6.16). Our DBSCAN Clustered LinearRegression model, like in the other clustering cases, fell quite a bit short of the other predictive algorithms both in terms of error and accuracy. LinearRegression can officially be ruled out as a best predictive model for our case - in no circumstance, baseline or clustered, did it perform to the level of other predictive models. We see the same centering of predictions for the data. This also supports our idea that the predictive algorithm influences outcome much more so than clustering (although clustering does enhance predictions in some cases).

Table 6.13 DBSCAN Linear Predictions. Again, we see centering around certain X and Y values

timestep_time	Cluster	Linear X Predicted	Linear Y Predicted
0	-1	1114.42778	1708.882616
1	-1	1123.531446	1709.523905
1	0	1109.717799	1783.348773
2	-1	1136.296035	1711.755934
2	0	1120.164436	1784.063295

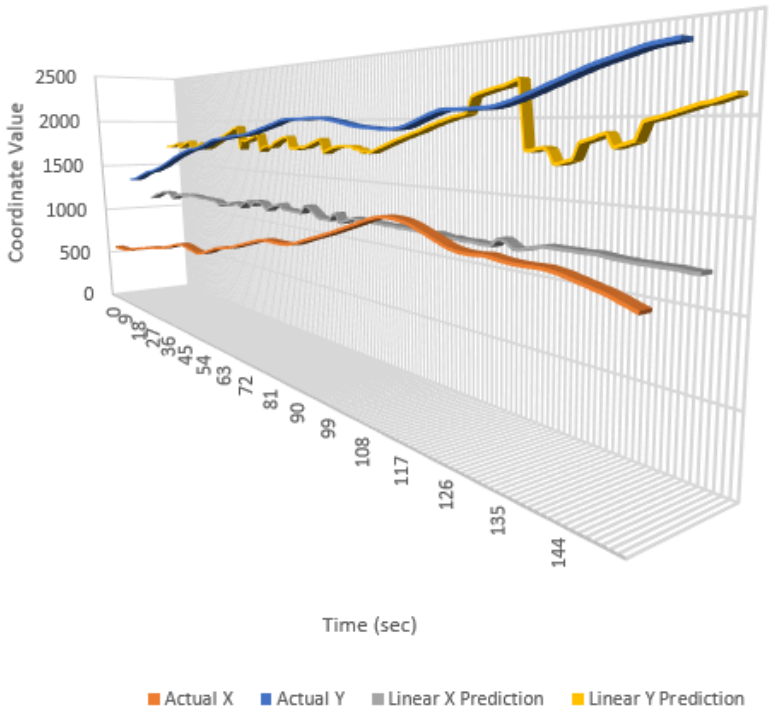


Figure 6.14 DBSCAN Actual vs Predicted for LinearRegression. We see a "Best Fit" line for each X and Y value

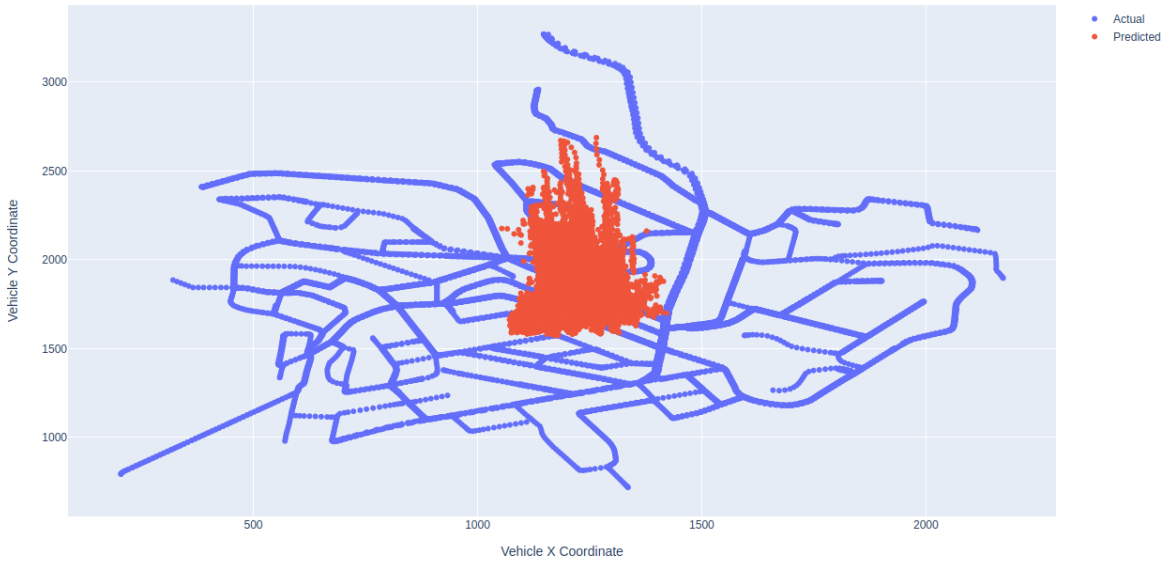


Figure 6.15 DBSCAN Actual vs Predicted for LinearRegression. The predictions are shown to be a middle point of all data points

6.4.2 DBSCAN KNeighbors Results

Our KNeighbors algorithm shows a somewhat accurate predictions with a relatively large error. Figure 6.16 shows, again, some very accurate predictions but a decent amount of error when predicting incorrectly. This is further ingeminated by our mapping of actuals and predicted values. Figure 6.17 shows many inaccurate predictions in the center of the map, while predictions become more accurate as data points find themselves on the edge of the map. The multi-layer approach with DBSCAN and KNeighbors did show relatively significant improvement [42, 43] over KNeighbors with other cluster (or non-clustering) methods. It is difficult to differentiate each clustering method's KNeighbors graph - in addition, across the board, KNeighbors does not perform at a high enough level to beat DecisionTree and therefore, the marginal gains ultimately become irrelevant.

Table 6.14 DBSCAN KNeighbors Predictions. Higher and lower coordinate values relative to the center are more accurate

timestep_time	Cluster	K-Nearest X Predicted	K-Nearest Y Predicted
0	-1	903.52	1442.228
1	-1	903.52	1442.228
1	0	2109.354	2172.206
2	-1	731.262	1392.694
2	0	2109.354	2172.206

6.4.3 DBSCAN DecisionTree

Lastly, our DecisionTree prediction appears highly accurate at 99.22%, and the plot in Figure 6.18 shows few mistakes as it did for other clustering methods. However, the error is slightly higher at 20.27 and accuracy slightly lower 99.22% relative to our non-clustered baseline of 20.02 and 99.24% respectively. As small gains can justify and validate a

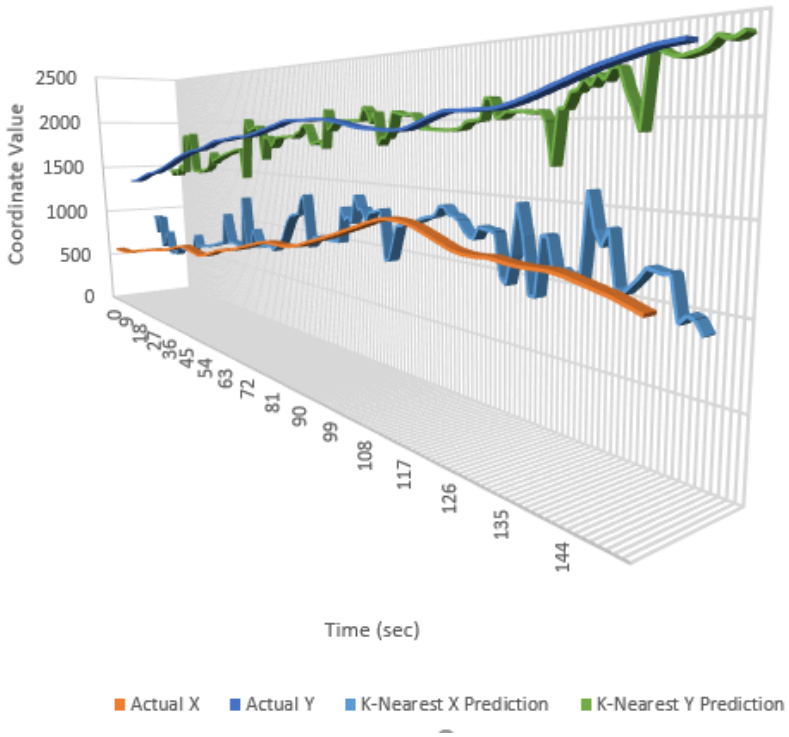


Figure 6.16 DBSCAN Actual vs Predicted for KNeighbors. A number of deviations from the actual can be seen

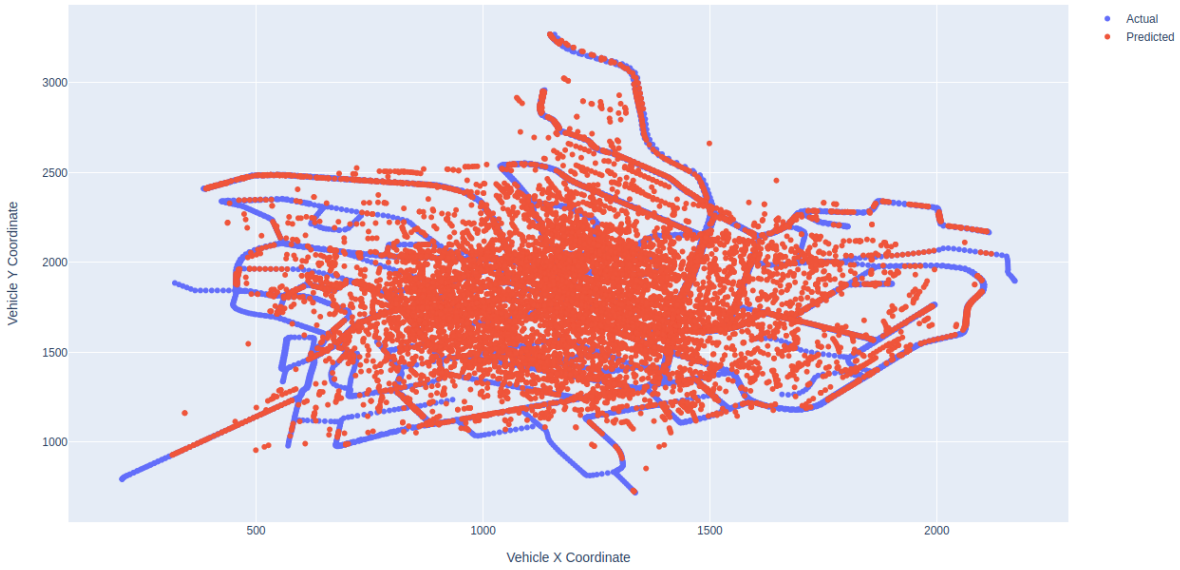


Figure 6.17 DBSCAN Actual vs Predicted for KNeighbors. K-Means has less accurate predictions in the center of the map

Table 6.15 DBSCAN DecisionTree Predictions. Results align with original data

timestep_time	Cluster	D Tree X Predicted	D Tree Y Predicted
0	-1	558.77	1336.89
1	-1	559.03	1338.38
1	0	2112.84	2170.77
2	-1	559.66	1342.02
2	0	2111.22	2171.44

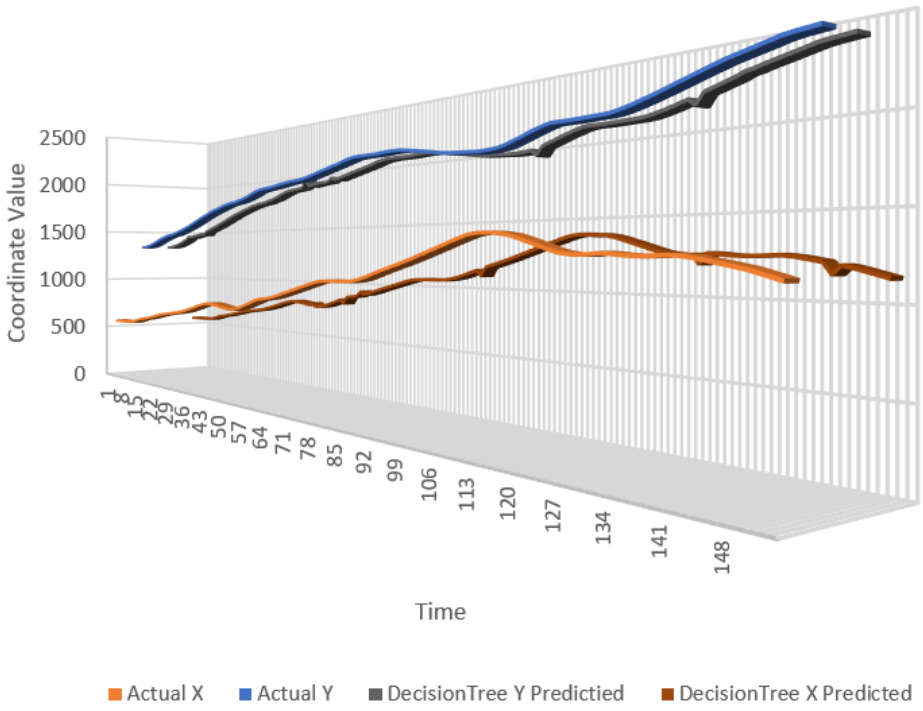
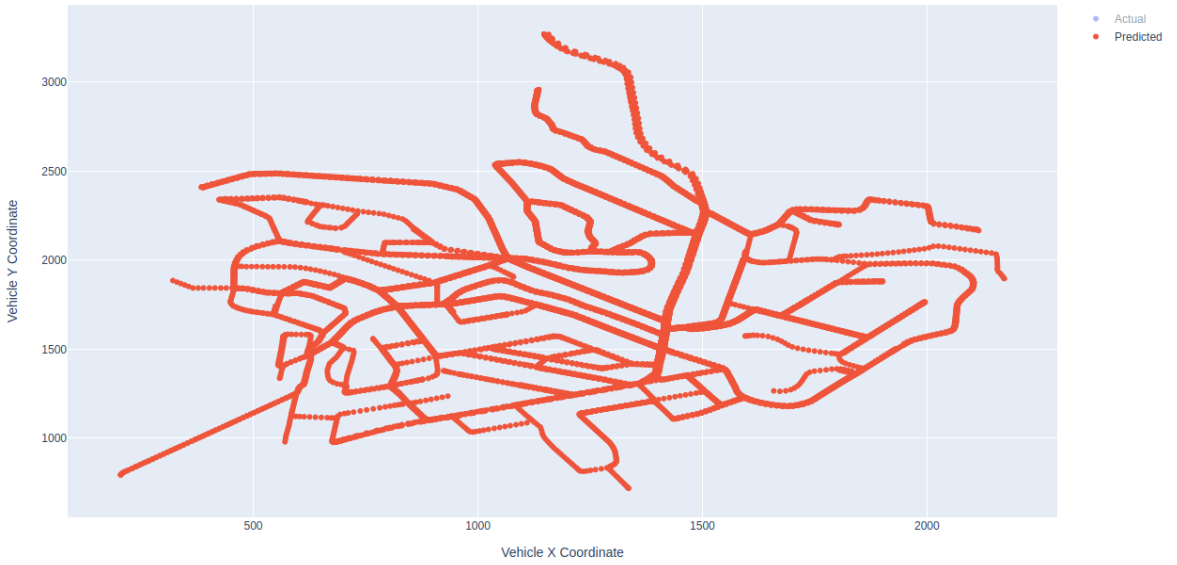


Figure 6.18 DBSCAN Actual vs Predicted for Decision Tree. There are a very few slightly off predictions

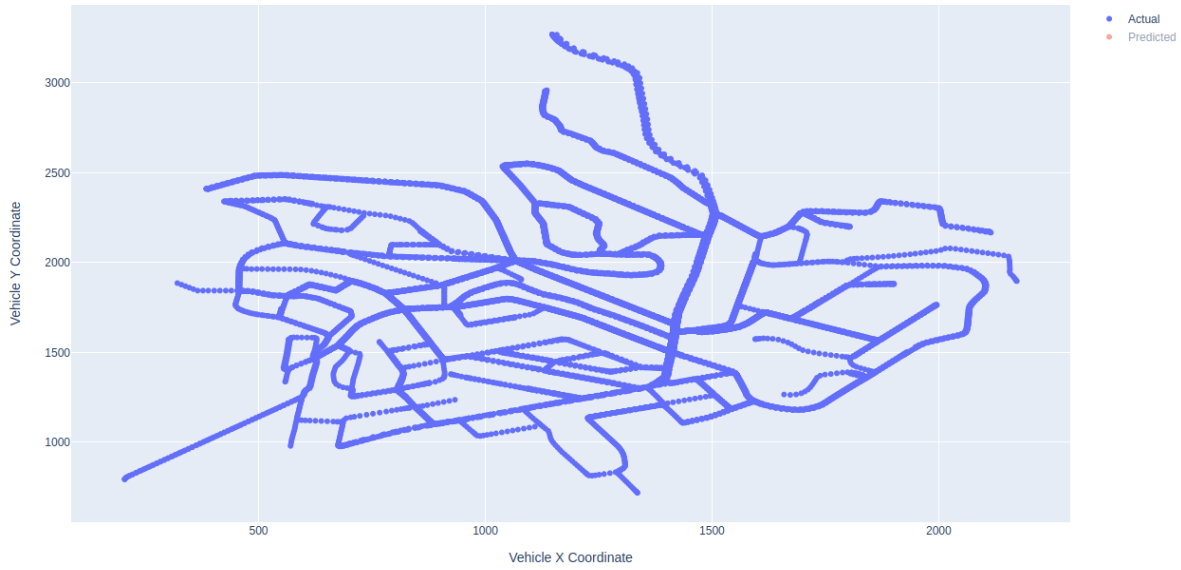
model, small losses can void a closely accurate prediction [42,43]. Although close, DBSCAN Clustering was unable to provide us with a *Cluster* variable that, when passed through prediction models, increased the reliability of the prediction.

Table 6.16 DBSCAN Summary. Our best predictive model, DecisionTree, just barely missed beating out our baseline of 99.24%

Predictive Model	Mean Absolute Error	Percentage Accuracy
DBSCAN LinearRegression()	299.70	88.53%
DBSCAN KNeighborsRegressor()	146.462	94.40%
DBSCAN DecisionTreeRegressor()	20.274	99.22%



(a) Predicted Values



(b) Actual Values

Figure 6.19 Predicted and Actual values plotted over our map. We see very little difference in the two meaning that our predictions were accurate

6.5 Application Against Threat Model

Comparing all the models and results of each, we were successfully able to build a multi-layer machine learning approach to improve predictions of vehicle locations - the

successful model was the combination of BIRCH Clusting with a DecisionTreeRegressor that achieved a 99.49% accuracy with an average absolute error of 13.32 coordinate points. This will be the model we can use to mitigate the risk associated with a replica and injection attack. We can use this model to identify these threats, and our predictions will make it very obvious (relative to the other models) that the network is under a type of threat.

An injection attack will occur when a vehicle's location is lost - or, X and Y for a given time period will equal 0. The vehicle will then be 'injected' back into the network, and using our high accuracy multi-layer BIRCH Clustered DecisionTree model, we can prevent any bad information from being propagated in the network when a vehicle returns. We have modeled this in Figure 6.20. It is clear that the vehicle dropped out of the network and returned twice, but our predictions remained true to actual values upon return. An injection attack would be much less obvious to visualize under a different model like LinearRegression where there are large portions of predictions very far from the actuals.

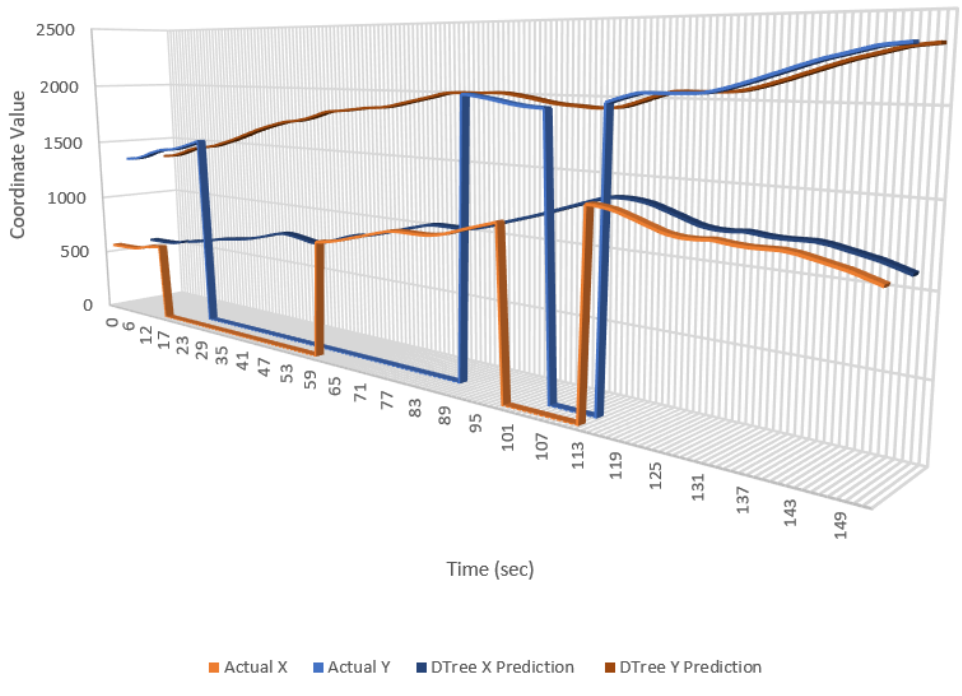


Figure 6.20 Injection Attack. Vehicle information is lost at *time=20* and *time=100*. The vehicle is injected back into the network and locations are already known

For replica attacks, the same vehicle will appear in 2 different locations at the same time. Our model can identify and prevent replica attacks shown in Figure 6.21. If we plot a replica attack on some of the other time based plots, for example any of the KNeighbors plots, it would be much more difficult to see these outliers. KNeighbors, in all clustering cases, had predictions with at least a few have quite a large error. In which case, these replica attacks would be much harder to spot, but our accurate BIRCH Clustered DecisionTree model shows these points as clearly inconsistent with the expected values.

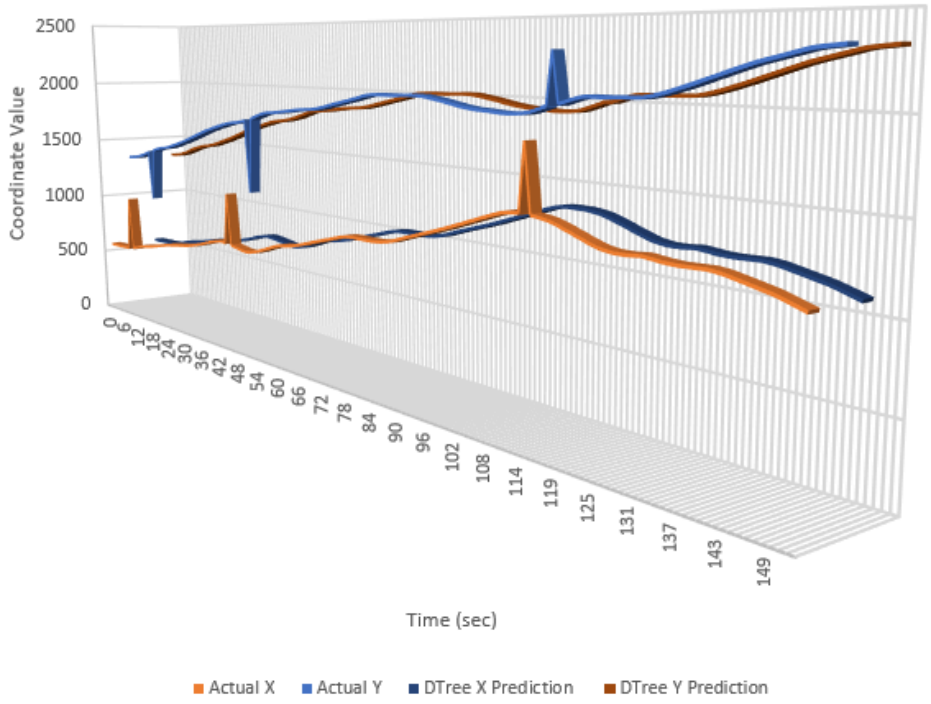


Figure 6.21 Replica Attack. Vehicle locations are incorrectly reported and are clearly oddities relative to expected values

CHAPTER 7

Conclusion

This chapter discusses our results in Section 7.1. We close with some future work ideas and shortcomings of this working in Section 7.2.

7.1 Closing Thoughts

Ultimately, we were able to create a multi-layer machine learning model to predict vehicle locations and prevent certain VANET threats. Moving through all the models, it became clear that the predictive model had much more of an impact than the clustering. However, clustering was still effective in ehancing predictions. BIRCH Clustering with a DecisionTree Regressor added a cluster feature that improved prediction accuracy by 0.25%, and in large data sets and machine learning cases, 0.25% is quite a jump. For example, if NHTSA estimates that 615,000 crashes per year can be prevented with enhanced vehicle to vehicle communications, a 0.25% improvement to the factors contributing to that 615,000 would result in preventing an additional 1,538 crashes per year. In addition to an increase in prediction accuracy, we were able to show how these predictive models could be used to prevent replica and injection attacks. Predicting vehicle locations with high accuracy can give governments (or other entities) more of an ability to prevent many different types of roadway hazards. Machine learning algorithms have slowly crept into our lives more and more, whether it be Facebook knowing what ads to show a person or pace makers telling

doctors when a heart attack, with mixed reviews depending on the application of machine learning. At this point however, machine learning and predictive analytics is a field that is here to stay and will continue to grow in the future. The applications will better lives because of the increased ability to make informed decisions about data that was inaccessible 10 or so years ago.

7.2 Future Work

7.2.1 Model Advancements

This work only compared a select few of available algorithms that could be applied to VANET data. Our predictions could have been improved by algorithms we did not implement. For clustering, an adaptive K-Means model could potentially be a great solution for the lack of possible predictive options in K-Means (DecisionTree and KNeighbors only have 3 clusters to learn on). Adaptive K-Means has the ability to recluster as data changes over time - so, vehicles can recreate clusters as they move through a simulation. In addition, we did not normalize the Euclidean distance for all the variables of K-Means. Normalizing Euclidean distances accounts for different units for variables. In our case, we compared a speed in meters per second to an angle variable in degree units. However, this does not apply to just K-Means. A mechanism for feature scaling needs to be applied to all the data to reduce bias in clustering and predictions. This would prevent algorithms from favoring certain variables for prediction. The lack of alignment in variable units can cause inaccuracies in clusters; non-similar items may actually end up being grouped together.

For our predictive models, DecisionTree was clearly the best predictive model in all cases and also was the only supervised learning algorithm examined. It would be worth while to look into how other supervised learning algorithms would aid in prediction; we did not examine these due to immense data preparation.

7.2.2 Simulation

Another shortcoming of this is work is the use of simulated traffic. BIRCH clustering was actually one of the slowest to run algorithms tested in this paper. It took roughly a minute to run relative to a second or two for K-Means and DBSCAN. In a real-world situation, that extra minute could be extremely costly. In addition, SUMO provided very limited variables. Some significantly more complex simulators, like Lyft's Project Level 5 data, provide a massive amount variables and vehicle movement data from real-world collected semi-autonomous vehicles. In relation to the simulator, we did not examine any anomaly-like situations. These would be situation like wrecks, heavy traffic, road closures, and other non-normal situations. It would be crucial to add these to our work and define how these situations would be modeled. Events like these occur every day and would very likely effect the clustering and ultimately the outcome of our predictions. Lastly, a mechanism to account for "near-miss" situations would help us accurately simulate real-world driving. On the road, different drivers have different tendencies of how close they are comfortable to another vehicle. How can we create a "safe zone" for every driver? How do sudden changes in speed or direction affect our predictions? Overall, we really only examined a non-normalized vehicle direction data set. To accurately model the real-world, someone would need to create many niche mechanisms for many different possible road scenarios and make sure data is accurately represented before clustering and predictions.

7.2.3 Data Storage

Lastly, data storage of any VANET is going to be a huge issue. We did not address data size much in this paper, but imagine the amount of data that would be collected if we could actually get all vehicles' information every second for a city. Within a week or so, you would have billions of data points. How do you address what gets stored? What gets

deleted? Also, there is a huge amount of processing power required to work with a data set that size.

REFERENCES

[1] "Is a v2v communication system worth it?" Jul 2019. [Online]. Available: https://daseuropeanautohaus.com/is-a-v2v-communication-system-worth-it/

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

[3] M. Roser and H. Ritchie, "Technological progress," *Our World in Data*, 2013, https://ourworldindata.org/technological-progress.

[4] R. R. Schaller, "Moore's law: past, present and future," *IEEE Spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[5] "Bigbelly smart waste platform for colleges universities," Oct 2019. [Online]. Available: https://bigbelly.com/solutions/campus/

[6] A. O Bang and P. Ramteke, "Manet: History,challenges and applications," 09 2013.

[7] B. Huber, "Behavioral model based trust management design for iot at scale," *Masters Theses and Doctoral Dissertations*, 08 2020.

[8] R. Khatoun and S. Zeadally, "Cybersecurity and privacy solutions in smart cities," *IEEE Communications Magazine*, vol. 55, no. 3, pp. 51–59, 2017.

[9] Anonymous, "Vehicle-to-vehicle communication," Dec 2019. [Online]. Available: https://www.nhtsa.gov/technology-innovation/vehicle-vehicle-communication

[10] V. Manjula and C. Chellappan, "The replication attack in wireless sensor networks: Analysis and defenses," in *Advances in Networks and Communications*, N. Meghanathan, B. K. Kaushik, and D. Nagamalai, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 169–178.

[11] Y. Xin and X. Feng, "Replica attack detection method for vehicular ad hoc networks with sequential trajectory segment," *International Journal of Distributed Sensor Networks*, vol. 15, p. 155014771982750, 02 2019.

[12] J. Grover, N. Prajapati, V. Laxmi, and M. Gaur, "Machine learning approach for multiple misbehavior detection in vanet," pp. 644–653, 2011.

[13] T. Khoshgoftaar, S. Nath, and S. Zhong, "Intrusion detection in wireless networks using clustering techniques with expert analysis," 2005.

[14] M. Slavik and I. Mahgoub, "Applying machine learning to the design of multi-hop broadcast protocols for vanet," *Wireless Communications and Mobile Computing Conference*, 2011. [Online]. Available: https://ieeexplore.ieee.org/document/5982799/

[15] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/

[16] S. Patterson, "Letting the machines decide," Jul 2010. [Online]. Available: https://www.wsj.com/articles/SB10001424052748703834604575365310813948080

[17] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[18] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *CoRR*, vol. abs/1203.6402, 2012. [Online]. Available: http://arxiv.org/abs/1203.6402

[19] M. Lima, B. Bogaz ZarpelÃ£o, L. D. H. Sampaio, J. Rodrigues, T. Abrao, and M. Junior, "Anomaly detection using baseline and k-means clustering," 09 2010.

[20] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," *SIGMOD Rec.*, vol. 25, no. 2, p. 103â114, Jun. 1996. [Online]. Available: https://doi.org/10.1145/235968.233324

[21] T. Zhang and U. Fayyad, "Birch: a new data clustering algorithm and its applications," *Data Min. Knowl. Disc*, pp. 141–182, 1997.

[22] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley, 1990.

[23] A. Ram, J. Sunita, A. Jalal, and K. Manoj, "A density based algorithm for discovering density varied clusters in large spatial databases," *International Journal of Computer Applications*, vol. 3, 06 2010.

[24] K. Sheridan, T. Puranik, E. Mangortey, O. Pinon, M. Kirby, and D. Mavris, "An application of dbscan clustering for flight anomaly detection during the approach phase," 01 2020.

[25] D. A. Freedman, *Statistical Models: Theory and Practice*, 2nd ed. Cambridge University Press, 2009.

[26] N. Ding, G. Tan, and W. Zhang, "Ia2p: Intrusion-tolerant malicious data injection attack analysis and processing in traffic flow data collection based on vanets," *International Journal of Distributed Sensor Networks*, vol. 12, no. 5, p. 5159739, 2016. [Online]. Available: https://doi.org/10.1155/2016/5159739

[27] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 683–697, 1992.

[28] C. Igel, N. Hansen, and S. Roth, "Covariance matrix adaptation for multi-objective optimization," *Evolutionary computation*, vol. 15, no. 1, p. 1â28, 2007. [Online]. Available: https://doi.org/10.1162/evco.2007.15.1.1

[29] R. Allmendinger, X. Li, and J. Branke, "Reference point-based particle swarm optimization using a steady-state approach," in *Simulated Evolution and Learning*, X. Li, M. Kirley, M. Zhang, D. Green, V. Ciesielski, H. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K. C. Tan, J. Branke, and Y. Shi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 200–209.

[30] C.-K. Liang and Y.-H. Lin, "A coverage optimization strategy for mobile wireless sensor networks based on genetic algorithm," *Proceedings of IEEE Internation Conference on Applied System Innovation 2018*, pp. 1272–1275, 2018.

[31] B. Bako, F. Kargl, E. Schoch, and M. Weber, "Advanced adaptive gossiping using 2-hop neighborhood information," 01 2008, pp. 5474–5479.

[32] N. Wisitpongphan, O. Tonguz, J. Parikh, P. Mudalige, F. Bai, and V. Sadekar, "Broadcast storm mitigation techniques in vehicular ad hoc networks," *Wireless Communications, IEEE*, vol. 14, pp. 84 – 94, 01 2008.

[33] X. Liu, Y. Liu, Y. Chen, and L. Hanzo, "Trajectory design and power control for multi-uav assisted wireless networks: A machine learning approach," 12 2018.

[34] P. Szczurek, B. Xu, J. Lin, and O. Wolfson, "Spatio-temporal information ranking in vanet applications." *IJNGC*, vol. 1, pp. 52–72, 01 2010.

[35] A. Galindo-Serrano and L. Giupponi, "Distributed q-learning for aggregated interference control in cognitive radio networks," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 4, pp. 1823–1834, 2010.

[36] M. Chen, W. Saad, C. Yin, and m. Debbah, "Echo state networks for proactive caching in cloud-based radio access networks with mobile users," *IEEE Transactions on Wireless Communications*, vol. PP, 07 2016.

[37] N. Taherkhani and S. Pierre, "Centralized and localized data congestion control strategy for vehicular ad hoc networks using a machine learning clustering algorithm," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, pp. 1–11, 04 2016.

[38] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ," https://www.openstreetmap.org, 2017.

[39] A. D. Helfrick and D. Cooper, William, *Modern electronic instrumentation and measurement techniques /*, na ed. New Delhi,: Prentice Hall,,, c1990.

[40] M. A. Syakur, B. K. Khotimah, E. M. S. Rochman, and B. D. Satoto, "Integration k-means clustering method and elbow method for identification of the best customer profile cluster," *IOP Conference Series: Materials Science and Engineering*, vol. 336, p. 012017, apr 2018. [Online]. Available: https://doi.org/10.1088%2F1757-899x%2F336%2F1%2F012017

[41] T. Kirasich, Kaitlin; Smith and B. Sadler, "Random forest vs logistic regression: Binary classification for heterogeneous datasets," *SMU Data Science Review*, vol. 336, p. 012017, 2018.

[42] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, Y. Lechevallier and G. Saporta, Eds. Heidelberg: Physica-Verlag HD, 2010, pp. 177–186.

[43] Fisher, "The netflix prize: Crowdsourcing to improve dvd recommendations," Oct 2015. [Online]. Available: https://digital.hbs.edu/platform-digit/submission/the-netflix-prize-crowdsourcing-to-improve-dvd-recommendations/

VITA

Sam was born and raised in Chattanooga, TN. After graduating from University of Tennessee, Knoxville in 2016, he began the first couple years of his professional career in Atlanta, GA. Eventually, he moved back to further his education by pursuing an Master's in Computer Science with University of Tennessee, Chattanooga. While working on his degree, he worked with UTC and Tennessee Valley Authority in various Python development and data science capacities. Sam intends to accept a full-time job with TVA in January 2021 as a Data Scientist. In his free time, he enjoys playing guitar, competitive road cycling, and running.