DRIVING BEHAVIOR CLASSIFICATION USING SMARTPHONE SENSOR DATA

by

Deniz Dikbıyık

B.S., Computer Education and Educational Technology, Boğaziçi University, 2020

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Software Engineering
Boğaziçi University

2023

ACKNOWLEDGEMENTS

I would like to thank my advisor for his guidance throughout my master's program. I would like to thank all my professors at Boğaziçi University for their great support during my education.

I give my deepest thanks to my family and friends for their great support and patience during my whole study. They always motivated and supported me during all of my hard times. Without their encouragement, I would not have been able to complete this journey.

I confirm that all of the content in the thesis is written by me. I always referred to the references if I got any inspiration during my writing. A scientific article from this thesis is under evaluation for publication. There might be conflicts with this thesis when my article is published.

ABSTRACT

DRIVING BEHAVIOR CLASSIFICATION USING SMARTPHONE SENSOR DATA

The need for driver behavior monitoring systems has increased due to the significant amount of accidents that are brought on by human mistakes. These systems have the potential to lower accident rates and increase overall road safety by offering real-time monitoring and analysis of driving behavior. Based on the data collected from passengers' smartphones, we propose a novel analysis framework for classifying driving behavior in this thesis.

Our mobile phone application was used to collect the data, which was then subjected to machine learning algorithms for processing. We utilized several Machine Learning (ML) classification techniques, with a particular emphasis on developing a Long Short Term Memory (LSTM) algorithm for increased accuracy and sequence-based prediction. The outcomes show how successfully the suggested method classifies driving behavior using the data obtained from smartphone applications.

After having a successful result with LSTM, instead of collecting all data from users into one area, we developed a federated learning algorithm to train and test each data on users' phones. The results of the study show that federated learning is useful for the classification of driver behavior and thus increases accuracy.

ÖZET

AKILLI TELEFON SENSÖR VERİLERİ KULLANILARAK SÜRÜŞ DAVRANIŞI SINIFLANDIRILMASI

İnsan hatasından kaynaklanan önemli miktarda kaza nedeniyle sürücü davranış izleme sistemlerine olan ihtiyaç artmıştır. Bu sistemler, sürüş davranışının gerçek zamanlı izlenmesini ve analizini sunarak kaza oranlarını düşürme ve genel yol güvenliğini artırma potansiyeline sahiptir. Yolcuların telefonlarından toplanan verilere dayanan çalışmamızda, sürüş davranışını sınıflandırmak için yeni bir analiz çerçevesi önerdik.

Veriler, akıllı telefonlara yüklenen, geliştirdiğimiz mobil uygulama kullanılarak toplandı ve Makine Öğrenimi (ML) algoritmalarını kullanarak işledik. Çalışmamız, doğruluk oranını artırmak ve sekans bazında sonuçlar çıkarmak için Uzun Kısa Süreli Bellek (LSTM) algoritması geliştirmeye özellikle vurgu yaparak, birkaç makine öğrenimi sınıflandırma tekniği kullandı. Sonuçlar, yöntemin akıllı telefon uygulamalarından elde edilen verileri kullanarak sürüş davranışını ne kadar başarılı bir şekilde sınıflandırdığını göstermektedir.

LSTM algoritmasının başarılı olması sonrasında, kullanıcılardan verileri alarak tek bir merkezde toplamadan, öğrenme ve analiz gerçekleştirebilmek amacıyla federe öğrenme algoritması geliştirdik. Çalışmamız federe öğrenme algoritmamızın sürüş davranışı sınıflandırılmasında başarı oranını da artırdığını göstermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

b	Bias vector
C1	Context unit
f_t	Forget gate
h_t	Output in related sequence
i_t	Input gate
$mean_x$	Mean value of x axis of the sensor data
$mean_y$	Mean value of y axis of the sensor data
$mean_z$	Mean value of z axis of the sensor data
o_t	Output gate
r_t	Reset gate in related sequence
$tanh$	Hyperbolic tangent function
W	Weight
x_t	Input in related sequence
x_{t-1}	Input in previous related sequence
x_{t+1}	Input in next related sequence
y_t	Another input in related sequence
y_{t-1}	Another input in previous related sequence
y_{t+1}	Another input in next related sequence
z_t	Update gate in related sequence
σ	Sigmoid function

LIST OF ACRONYMS/ABBREVIATIONS

A	Accelerometer
Adam	Adaptive Moment Estimation
ANN	Artificial Neural Networks
CNN	Convolutional Neural Networks
DNN	Deep Neural Networks
EEG	Electroencephalographic
EOG	Electro-oculographic
FCN	Fully Convolutional Networks
G	Gyroscope
GRU	Gated Recurrent Units
GPU	Graphical Processing Unit
IoT	Internet of Things
IoV	Internet of Vehicles
LSTM	Long Short Term Memory
MAX	Minimum
MIN	Maximum
ML	Machine Learning
NHTSA	National Highway Traffic Safety Administration
NMVCCS	National Motor Vehicle Crash Causation Survey
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Networks
STD	Standard Deviation
VAR	Variance
X	X axis
XYZ	X, Y and Z axes
Y	Y axis
Z	Z axis

1. INTRODUCTION

1.1. Problem Definition

Modern technology has given people many possibilities like changing location easily [1]. Transportation is very important for people to socialize. It also forms the basis of economic activities [2]. While vehicles support people to travel on the route they want, they also cause problems such as traffic congestion and public transportation problems. Most of these problems are caused by driver behavior [3]. According to National Motor Vehicle Crash Causation Survey (NMVCCS) conducted in 2005 by the US National Highway Traffic Safety Administration (NHTSA), it is seen that 94 % of accidents are caused by drivers. Vehicles, environment, and some other reasons play minor roles in accidents as seen in Table 1.1 [4].

Table 1.1. Critical reasons of accidents [4].

Critical Reason Attributed to	**Estimated**	
	Number	**Percentage**
Drivers	2,046,000	94% ±2.2%
Vehicles	44,000	2% ±0.7%
Environment	52,000	2% ±1.3%
Unknown Critical Reasons	47,000	2% ±1.4%
Total	2,189,000	100%

It is important to analyze driver performance to better understand driver behavior and find solutions to problems. Since the 1960s, many studies have aimed to find a solution to this issue [3]. Parameters such as speed, distance, and acceleration play an important role in determining driver behavior [1]. Road safety has also been suggested to be an important factor in driver behavior. Apart from the road effect, aggressive

drivers also exhibit unsafe driving styles. Daily influences such as demographic information, personality, level of aggression, decision-making ability, experience, alcohol, and stress also affect driving style as individual factors. Furthermore, environmental factors that affect road safety are the time of day of driving, pedestrians, lighting, other drivers, and weather conditions [5].

1.2. Proposed Method and Contributions

The main contribution of our study to the literature is that we classified the driver behavior with the data collected from our mobile application installed on the smartphones of the passengers. Similar to previous studies, we also suggested using machine learning algorithms. First of all, we studied the existing Machine Learning (ML) classification algorithms which are Logistic Regression, Support Vector Machine, K-Nearest Neighbors, Decision Tree, Random Forest, and Naive Bayes. After comparing the accuracy rates of the existing machine learning classification algorithms, we obtained higher accuracy results with the Long Shot Term Memory (LSTM) algorithm.

Since data security has become a very important issue, after developing the LSTM algorithm, we worked on the federated learning method to be able to classify driving behavior without collecting data from passengers' mobile phones in a central location. Thanks to federated learning algorithms, separate machine learning models can be trained with different data groups, and a global model can be generated from these separate models to be used for classification.

In our study, we used the data obtained from the accelerometer and gyroscope sensors on the smartphones of the passengers. The accelerometer sensor is used to detect the capacitance change of a moving mass and automatically detects the acceleration. Gyroscopes are devices that can detect angular velocity during rotation on a frame [6]. Thanks to the method we propose, risky driving can be detected when it takes place and precautions can be taken against dangerous situations.

1.3. Thesis Organization

The thesis organization is as follows. The following Chapter 2 explains the studies done before in the literature and what is needed to be improved with a comparative analysis. Chapter 3 provides the details of machine learning and deep learning concepts. Chapter 4 starts with the description of the data that we collected via the mobile application we developed and continues with how we process it before the experiments. Chapter 5 presents the architecture of the algorithms we use. Chapter 6 gives the results of each implementation for the experiment. Chapter 7 provides an analysis of our results. Chapter 8 concludes the thesis and gives a direction for future work.

2. LITERATURE SURVEY

2.1. Parameters Used for Driving Behavior Detection

In previous studies conducted to determine the parameters that give clues about the driver's behavior, it was understood that the speed varies, so they worked on a road tracking method. The predictive control model was used to have a path estimate of the driver's coordinates. Decision-making and time delays as a result of behavior were also analyzed [3]. Driving analysis was also integrated with autonomous vehicles, as driving style is believed to affect energy consumption. The most common factors found to be analyzed in the study were speed, acceleration, and pedal positions. Classification of the detected data has been a common way of labeling behavioral style [5].

There are many studies done with mobile applications in terms of sensor data usage. These apps use the accelerometer, gyroscope, position, gravity, magnetometer, and orientation sensors to collect data only from the driver's phone [1, 6]. In a study conducted only with mobile device accelerometer and gyroscope sensor data, the importance of abnormal acceleration, steering management, and vehicle skidding was mentioned. In that study, it was concluded that these sensor data can indicate abnormal behaviors and are successful in classifying normal or risky behaviors [7]. There have also been studies in which smartphone location information was evaluated together with acceleration detection. In one example, passengers were also mentioned, but it was aimed to inform the traveling passengers only in case of abnormal acceleration of the vehicle [8].

2.2. Methods Used for Driving Behavior Classification

Detection and vehicle data recording were utilized in monitoring driver behavior using smartphones [9]. Classification according to rules, model development, and the use of machine learning algorithms have been the common methods in the literature

for intelligent driving analysis observations [5]. Behavior detection has also been used to assist the driver with recording tools, video cameras, and other safety measures to improve intelligent transportation systems. Real-time and non-real-time analysis methods were chosen to finalize the collected data. Vehicle-mounted cameras enabled tracking eye movements and facial expressions for real-time control. In these studies, mobile phone sensors of the driver were also used, and neural network algorithms were developed to analyze them. Additional questionnaires to the same techniques were also included but resulted in a non-real-time classification [10]. For smart systems, an example of a method using a dynamic bayesian network that generates alarms to the driver, and the drivers of the surrounding vehicles was also available [11].

Once again, different analysis techniques classified according to the data source, tasks, and models for smart transportation systems and driver behavior analysis were mentioned in the literature. Smartphone sensor records, radar simulators, and multimedia collected from the driving experience were used as data inputs. Detection, identification, and estimation were the tasks reserved for this process. Machine learning algorithms such as deep learning, fuzzy logic, decision tree, bayesian, and regression were used for inference. Some classification labels such as careful, safe, normal, aggressive, risky, dangerous, moderate, calm, and environmentally friendly were mentioned for the purpose of labeling driver behavior. Some classifications have been made according to the security range [12].

The study in which the complex event processing method was applied focused on the use of the Internet of Things in driver behavior detection and the effect of cloud systems on data storage and processing. Smartphone and vehicle sensor data were used with the foresight that it would be easier to analyze the data instantly in real-time. The minimum, maximum, quartile, and median of the data were used to detect outliers. Emphasis was placed on finding the best algorithm in terms of accuracy, precision, recall, f-measure, average, execution time, and resource consumption [13].

Driver behavior classification has been tested with many different machine learning and technical algorithms [14]. However, there is a lack of large data sets to train algorithms because these studies were often done in simulated environments or with limited data collected. Since the algorithms used in the studies are generally machine learning, it is important to train with a large data set to obtain successful results. It is recommended to have a public data set for all studies and to benefit from this data in the relevant studies [15].

After the method decision and data collection, deciding which algorithm to use has also great importance in studies. In general, the machine learning algorithms preferred were logistic regression, neural networks, gradient boosting classifiers, random forest classifiers [16, 17], coordinated vehicular networks [18], one-dimensional convolutional neural networks [19], deep convolutional neural networks [20], deep learning methods [21, 22] and long short term memory algorithm [23-31]. Studies that apply machine learning and deep learning methods together to achieve highly successful results had been more successful [32, 33].

2.3. Supporting and Motivating Driver

Since safe driving motivation is an important issue, gamification has been among the methods used in this field. Drivers were rewarded by the detection application among their social networks according to their results. The data collected from the application were clustered with the K-means algorithm, then analyzed with an application developed in Python and finally connected with the SQLite database. Users were rated to be rewarded considering weather conditions [34]. Healthy driving systems have been developed for drivers' use on their mobile devices. These systems obtained information about the road condition by taking the accelerometer sensor data and were scored by the algorithm to show the results according to the thresholds [35]. The dissemination of driver behaviors and the help of wearable devices in safe driving studies were also mentioned. The eye states and yawns of the drivers were also taken into consideration in the studies. Although the success rates of the methods were not given in tables in

order, it was mentioned that hybrid applications give results with a high success rate and are applicable [36].

2.4. Data Privacy Perspective

The Internet of Vehicles (IoV) term which is derived from the Internet of Things (IoT) term was brought to the literature for data privacy. The term was mentioned with Federated Learning which enables training a global model without collecting and combining all the data from all drivers [37]. In Federated Learning, data are not loaded to the server totally, yet the model is trained with sessions separately, but the performance of the analysis still increases. While decreasing the requirements of communication, limited computing resources on vehicular edge devices can be used properly [38]. To assist drivers with stress detection and behavior classification, vehicular network computing for edges can be managed even during increased demand [39]. In the end, thanks to Federated Learning, both privacy of the personal data and the computing performance of the devices can be controlled easily [38].

2.5. Comparative Analysis of Related Works

To have a deep understanding and find what is lacking in the literature, we analyzed existing driving behavior analysis studies with machine learning and deep learning methods below. The summary look of the analysis can be seen in Table 2.1.

Table 2.1. Evaluation table of selected driving behavior classification studies.

Study	**Data From**	**Real Data**	**Method**
Ghandour et al. [16]	Driver	Yes (from literature)	ML
Di Giacomo et al. [17]	Driver	Yes (from literature)	ML
Shahverdy et al. [19, 20]	Driver	Yes (scenario based)	CNN
Al-Hussein et al. [21]	Driver	Yes	DNN, RNN, CNN
Mumcuoglu et al. [23]	Driver	No (simulation)	LSTM
Cura et al. [24]	Driver	Yes (scenario based)	LSTM, CNN
Kadri et al. [25]	Driver	Yes (from literature)	LSTM
Saleh et al. [26]	Driver	Yes (from literature)	LSTM
Kouchak and Gaffar [27]	Driver	No (simulation)	LSTM
Zhang et al. [28]	Driver	Yes (from literature)	LSTM, CNN
Khodairy and Abosamra [29]	Driver	Yes (from literature)	LSTM
Cojocaru et al. [30, 31]	Driver	Yes	LSTM, CNN
Huang et al. [32]	Driver	Yes (from literature)	CNN
Brahim et al. [33]	Driver	No (simulation)	LSTM, CNN
Chhabra et al. [37]	Driver	Yes	LSTM, CNN
Doshi and Yilmaz [38]	Driver	Yes (from literature)	CNN
Our Study	Passenger	Yes	ML, LSTM

Ghandour et al. [16] implemented the study with real existing data which was collected from six drivers and vehicles. They classified driving behavior as normal, aggressive, and drowsy. Before applying the methods they use, they randomized the data. Places the data was collected were motorway roads and secondary roads. From two aspects of features, they analyzed the data. From a lane detection perspective, timestamp, car position, car orientation, lane width, and driver's state (normal, aggressive, drowsy) features were grouped. From a traffic status perspective, timestamp, distance to the nearest car, time until collision with the nearest car, number of vehicles on the lane, speed in GPS, and driver's state features were grouped. They applied logistic

regression, artificial neural networks, gradient boosting, and random forest algorithms to analyze the data. As a result, the gradient boosting algorithm was found as the best-performing algorithm from both perspectives. The accuracy of the best algorithm was 60 % for lane detection and 67 % for traffic status.

Di Giacomo et al. [17] used real data in their study that was collected from control and measurement units of vehicles. They proposed learning and predicting steps for the flow and used Weka machine learning software to process their data. Accelerator pedal value, the torque of friction, fuel consumption, and air pressure taken in were some of the features considered. Also, engine soaking time, long-term fuel trim bank, transmission oil temperature, and steering wheel speed were the discriminatory features. They resulted that the algorithms they used in Weka software gave successful analysis, but they suggested that other classification machine learning algorithms should also be tested on the data for future work.

Shahverdy et al. [19] defined a scenario for drivers to behave and recorded three participants to collect real data. They analyzed vehicle acceleration, road slope, speed, steering angle, road curve, and speed limit of the road features. The data was labeled as safe, distracted, aggressive, drunk, and drowsy. To standardize the data, they used the scikit-learn library for standard scalar and windowed the collected data for time series to apply the one-dimensional Convolutional Neural Network (CNN) algorithm. The study resulted in 99.99 % accuracy and low computational complexity. Shahverdy et al. [20] also tested deep convolutional neural networks algorithm in another study with the same data and reached a 99.99 % accuracy rate. In their papers, they also emphasized that Gated Recurrent Units (GRU) and Long Short Term Memory (LSTM) algorithms gave nearly 95 % accuracy while Simple Recurrent Neural Network (RNN) gave approximately 70 %.

Al-Hussein et al. [21] collected their data from 30 participants which included speed, acceleration, deceleration, distance, and yaw steering columns. They divided the data into two categories safe and aggressive. After applying Deep Neural Net-

works (DNN), Recurrent Neural Networks (RNN), and Convolutional Neural Networks (CNN), they resulted that CNN gave the highest success with 96.1 % accuracy.

Mumcuoglu et al. [23] analyzed truck driver behavior with Long Short Term Memory (LSTM) algorithm according to acceleration limits. They used the Truck-Maker vehicle simulation application to generate their data. Longitudinal and lateral acceleration, engine, and vehicle speed, throttle, and pitch angle were the data inputs to classify. They windowed their data for 30 seconds and applied the LSTM algorithm. As a result, they got 74.7 % accuracy on the test.

Cura et al. [24] collected their data in a special area with real drivers, but the drivers applied scenario-based driving styles. They analyzed the data with engine speed pedaling, deceleration, corner turn, and lane change attempts measurements. The labels they worked for were aggressive, mild, and gentle driving behaviors. After windowing the data for 3.5 seconds, they applied both Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM) algorithms. They analyzed their labels only for 2 with different combinations. They explained the aim of using the LSTM algorithm as having time series data. They got at most 93.1 % accuracy in the test for both of the algorithms, but this accuracy was for aggressive or gentle classification.

Kadri et al. [25] used a real-time data set from the literature in their study. The data was labeled as normal, aggressive, and drowsy. The measurements they analyzed were acceleration on the x, y, and z-axis, roll, pitch, and yaw angles, and vehicle speed from the GPS sensor. They proposed a new stacked LSTM model with separate memories and a two-layer cell. They preferred to use the LSTM algorithm because they had time series data. The algorithm resulted in high performance with 97 % accuracy.

Saleh et al. [26] preferred to analyze driving behavior with LSTM data, but again preferred to work on a data set from the literature. Their measurements were again acceleration on the x, y, and z-axis, roll, pitch, and yaw angles, and vehicle speed from

a GPS sensor, but they also included a camera sensor to detect distance with other cars and the number of cars on the road. They classified data labels as normal, aggressive, and drowsy. Even if they tried different algorithms such as Multi-layer Perceptron, they resulted that LSTM gives higher results, and they contributed to the literature with 10 % more accuracy.

Kouchak and Gaffar [27] studied the LSTM algorithm with the data collected from the DS-600 simulator for Drive Safety Research. The was data collected from 35 participants and 45-minute recordings were analyzed with LSTM and Bidirectional LSTM Network. As measurements, they collected distraction status, driver error count, response time, and navigation steps. Their bidirectional LSTM algorithm resulted in higher performance than LSTM.

Zhang et al. [28] used a Multi-Scale Convolutional Neural Network algorithm to extract multi-scale temporal and spatial characteristics and Bi-directional Long Short Term Memory algorithm that gives a better performance for text analysis in their study to recognize driving behavior. For the study, they worked on a public data set that included coordinates and the status of driving as accelerating and decelerating. Even if the data set included information about vehicles, riders, and pedestrians, they focused mostly on the data from vehicles. They got 90.85 % accuracy with a combination of their methods.

Khodairy and Abosamra [29] focused on the labels normal, drowsy, and aggressive for driving behavior. They used the accelerometer, gyroscope, GPS data, and vehicle detection with a video-camera sensor from the DriveSafe application. They used a data set from the literature that can list the data given above. They both tried to classify the data with 2 and 3 labels with their LSTM algorithm model in different experiments. 3 label classifications gave the highest accuracy with 99.47 % as a result.

Cojocaru et al. [30, 31] focused on slow, normal, and aggressive labels to classify driving behavior. After analyzing and finding that the data sets in the literature were

not enough, not well labeled, and not collected exactly in real situations, they collected their data set from the driver. They used LSTM and CNN methods in two different studies. They classified their data both with 2 labels and 3 labels. As a result, in their first study, they reached up to 63.70 % accuracy with 3 labels and 79.50 % accuracy with 2 labels. In their second study, they improved the result by up to 91.94 % accuracy.

Huang et al. [32] used a data set from the literature to detect driver behavior. Their data included many labels as normal driving, texting, calling, playing with the radio, reaching behind, drinking, makeup - hair, and talking with passengers. They processed the images of the driver to classify the behavior. Their deep learning model with CNN achieved an accuracy of 94.72 %.

Brahim et al. [33] collected their data from the Carla simulator with the generation of different scenarios. They simulated different weather conditions and roads with normal, intermediate, aggressive, and dangerous driving styles. They implemented the LSTM algorithm in three different types alone, with Convolutional Neural Network and with Fully Convolutional Network (FCN) to classify the steering and speed data. The best accuracy they got from LSTM with a combination of Convolutional Networks was 79 %.

Chhabra et al. [37] collected smartphone data from accelerometer and gyroscope sensors and labeled them as safe, unsafe, or fatigue driving. Using CNN and LSTM algorithms, they classified the behavior and also emphasized on the term Internet of Vehicles (IoV) aiming for Federated Learning which is derived from the Internet of Things (IoT) for model training without collecting data from users. Their accuracy reached up to 89 % with 5 clients. It was seen that the experiment with 20 clients got approximately 87 % accuracy.

Doshi and Yilmaz [38] used 2 different data sets from the literature for benchmarking. While using the CNN algorithm, they combined their study with Federated Learning to compare the results. The data set they used included only images and

video captured for driver view where the drivers behaved in the given scenarios. Even if they did not share their decentralized algorithm results with federated learning, they concluded that the models they applied gave similar success rates and decentralized model could be used successfully.

2.6. Discussion

When we compare the related works, firstly, we see that collecting primary and real-life data are necessary in the literature. Existing studies generally referred to similar data sets which are especially UAH-DriveSet [25, 26, 29], or generated their data from simulators [23, 27, 33]. In real data collection processes, they included the scenario they created that is driven by the driver. That is why, collecting real-world data is vital to contribute to the literature.

We also realized that studies were conducted on the data gathered from the driver and the vehicle. To have a better understanding of passenger safety, we propose to collect data from passengers' smartphones. Doing so, we can also collect genuinely voted labels from the passengers and do not need to classify the data by designed rules. As we use mobile phones, we collect accelerometer and gyroscope sensor data. We did not include GPS since the path was specific.

For the method selection, simple Recurrent Neural Network (RNN) was found to be a less accurately resulting algorithm [19, 20], but Long Short Term Memory (LSTM) and Convolutional Neural Networks (CNN) were among the best-performing algorithms [24, 28, 33]. It was explained that CNN is performing better with image and video processing [21, 33], but for the data of time series recorded in tables, LSTM gives effective results [24, 28]. The explanation of the above-mentioned algorithms is given in Chapter 3. In the studies, it was also preferred and suggested to compare the results with classical machine learning algorithms [16, 17, 33]. When they analyzed the results, deep learning was found to get high performance among algorithms. We also propose to compare results with existing machine learning algorithms. To reduce

communication requirements and increase the success rate of the algorithms with small-session training, the Federated Learning model was also suggested in the literature to be used in driving behavior classification [37, 38]. In this study, we also propose to train a global model with different model weights without giving the whole data set to the LSTM algorithm.

2.6.1. Passenger Perspective

Passenger safety is important and it has not been studied before. It should be analyzed because thousands of passengers die every year [40]. Detection of how driving behavior affects passengers should be well understood to ensure safety of the passengers as numerous accidents are reported to occur while passengers are complaining to the driver. Passengers tend to inform drivers, talk and even complain about the driving experience [41]. Even if this distraction causes accidents, such warnings do not improve driving behavior because they generally make no difference to drivers [40]. If there can be a detection system that automatically classifies the driving experience, passengers can know that their safety is detected in a central mechanism. Therefore, the detection system from passengers' perspective can be used by local authorities to ensure that public transportation drivers are driving safely. Furthermore, the general directorate of highways can detect risky path directions and improve them. To reach a standardized education, many students go to schools that are not near their homes. Because they commute by school bus, passenger safety detection can be used by children and parents as well. Even if the parents wonder about the school bus atmosphere, they also focus on the road length and driving behavior. They assume that school bus drivers are speeding on rural roads [42]. As a result, to know about their child's safety, parents can install the driving behavior detection application on their child's smartphone.

3. BACKGROUND

3.1. Machine Learning

Machine learning algorithms are developed to reduce human work in different areas. These algorithms are based on artificial intelligence and learn from their errors. When data are given, they can analyze it according to their needs [43, 44].

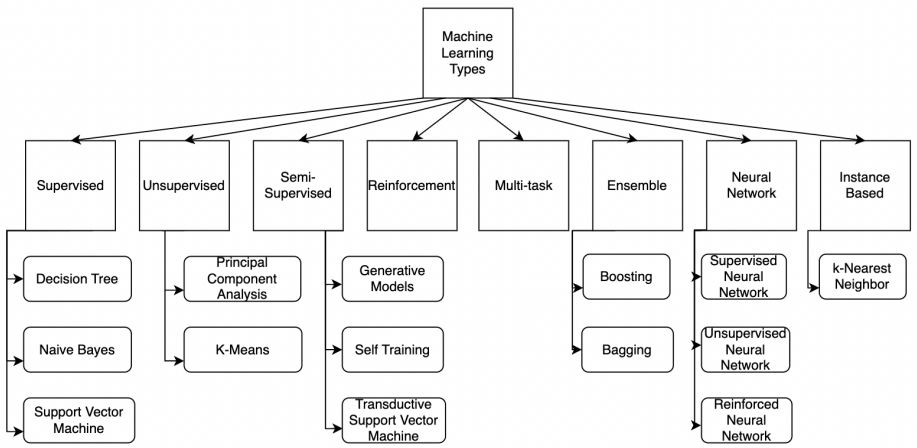

Figure 3.1. Machine Learning Types and Algorithms, Adapted from [45].

When the output of the data is given with labels mapped with inputs, it is called supervised learning. On the other hand, the unsupervised learning occurs when the labels are not given [45]. In addition to the supervised and unsupervised algorithm types, there is another term as reinforcement learning, which is mostly preferred in the Internet of Things and Robotics area. After getting some sensor data, it decides the action for the next step [46].

These algorithms are used in a wide range such as image recognition, fall detection, text classification, bio-informatics, and voice recognition [44, 46]. While machine learning algorithms solve different problems, the most common areas they are used are:

- Classification,
- Regression,
- Clustering,
- Association.

Classification algorithms only focus on the data to classify it. The regression method predicts numerical values, while the classification algorithm is interested in categories. Clustering differs from the others with grouping ideas. It groups the data by some features. Finally, the association algorithm works with rules [46].

3.2. Machine Learning Models

The machine learning process includes three steps: train, preview, and action. During the training step, the data is given to the model without any guidance. Secondly, preview results with data relations to make predictions. In the action stage, the model is guided with feedback on correct or incorrect results [46]. Linear Regression, Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbors, Support Vector Machine, K-Means Clustering, and Naive Bayes are among the machine learning algorithms widely used [43, 44].

3.2.1. Linear Regression

Linear regression is based on a straight-line equation. It estimates the relationship between independent and dependent variables [47].

3.2.2. Logistic Regression

Even if it is similar to linear regression for deciding dependent and independent variables, logistic regression is a type for classification because it can predict probabilities. It classifies as 1 or 0 for conditions like true or false [47].

3.2.3. Decision Tree

The decision tree splits the data according to the question and feature relations. It is a type of supervised learning. To predict, this algorithm generates a decision tree to reduce randomness [47].

3.2.4. Random Forest

The random forest is similar to the decision tree, but it generates multiple trees. This algorithm works with the idea voted the most by many trees. Then the counts of the results are summed to conclude [47].

3.2.5. K-Nearest Neighbors

K-Nearest neighbor is a classification algorithm according to the nearest data point. The new item is assigned to the most neighbor-assigned class [47]. For nearest neighbor detection, Euclidean, Minkowski, Manhattan or Hamming distance equations can be used [48].

3.2.6. Support Vector Machine

Support vector machine is a type of supervised learning, and is useful for both regression and classification problems. To decide for different classes, it generates a decision boundary or a hyperplane [47, 48].

3.2.7. K-Means Clustering

As a type of unsupervised learning algorithm, K-means clustering works with a K number of clusters to have similar points in it. These clusters are randomly selected and adjusted until being highly distinct [47].

3.2.8. Naive Bayes

Naive Bayes is based on a probabilistic model related to Bayes theorem which claims that all of the features are independent. It predicts the probability to decide which class an item belongs to [47].

3.3. Deep Learning

Thanks to high accuracy, deep learning algorithms are found valuable in literature as a type of machine learning. With connected layers, deep learning algorithms should be trained with big data sets. The idea behind it is the cerebral cortex as human brains work [49]. To have a similar process for artificial neurons, perceptrons are defined by Frank Rosenblatt in the 1950s. Perceptrons get several inputs and produce a single output [50]. Even if perceptrons cannot process the input as the human brain does, they give some weights to inputs and conclude output accordingly.

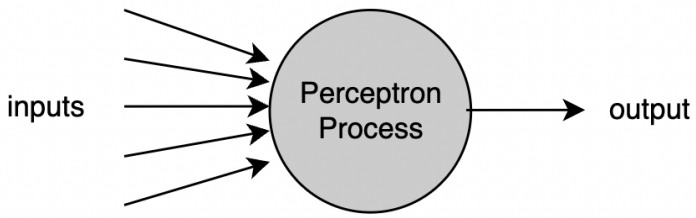

Figure 3.2. Perceptron Process, Adapted from [49].

3.4. Differences of Machine Learning and Deep Learning

Deep learning is a part of machine learning. Only the process is different for these two types of algorithms. Basic machine learning models still require some human involvement even if they do get better at doing their respective tasks when new data are added. On the other hand, a deep learning algorithm may evaluate the accuracy of a prediction using a neural network without assistance of a person. Deep learning organizes algorithms in layers to create an artificial neural network that can learn and decide in contrast to machine learning, which employs algorithms to parse data to learn and decide [51].

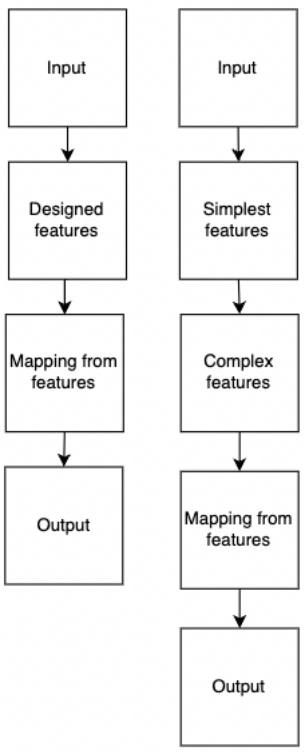

Figure 3.3. Differences in the Flow of Machine Learning and Deep Learning, Adapted from [52].

Deep learning networks require more computational power because they need more data to work better. Therefore, while machine learning algorithms can work with

what the CPU can provide, deep learning algorithms work better with a graphical processing unit (GPU), which has hundreds of cores as opposed to a CPU [53].

Deep learning algorithms also perform better when more data input is given. Machine learning algorithms can perform well with small data sets as well [53].

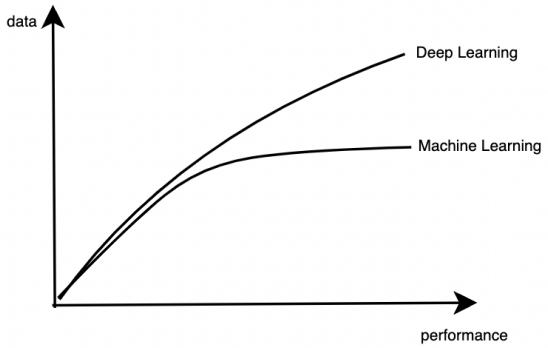

Figure 3.4. The Change in the Performance with Amount of Data, Adapted from [53].

In the industry, machine learning algorithms are used more commonly. Deep learning algorithms are not always found reliable to implement in the industry. Areas like banking and stock prediction mostly choose machine learning algorithms such as linear regression, decision trees, and random forests. In general, both machine learning and deep learning algorithms are used in medicine, text and image interpretation, and natural language processing [53].

3.5. Deep Learning Models

3.5.1. Neural Networks

There are several nodes in an artificial neural network. In Figure 3.5, every single circle represents a node for an artificial neuron, and arrows are connections [54].

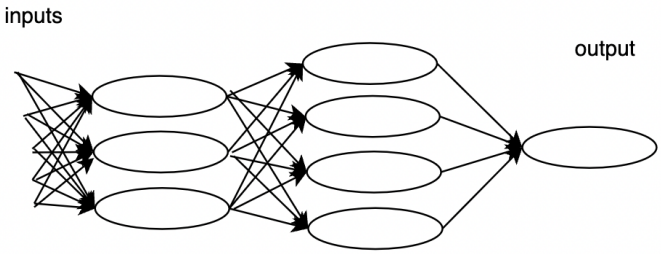

Figure 3.5. Neural Networks Layer Process, Adapted from [50].

There are three types of layers: input, hidden, and output. The hidden layer connects the input and output layers [54]. With the data from sensors or the files, input layer neurons gather the data to process. After the process is finished, the output layer shares the result with the outside [55]. Hidden layers implement a continuous data process and accuracy increases with the improvement. Neural network depth changes in accordance with the layer number [54]. The process mentioned here is the weight given to the features which enter as inputs, and the summation of them to conclude a vector representing how features affect the result with their weights [55].

3.5.2. Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) is widely used for image processing or document classification that is considered as an image. Without CNN, fully connected neural networks should take each pixel of an image as input for a neuron which means having as many neurons as the number of pixels on each side equals. With CNN, we can give pixel intensity as input instead of a multiplication of pixels. Here, input pixels

enter the hidden layer neurons. We can get a small part of the picture with the area of that pixel [50].

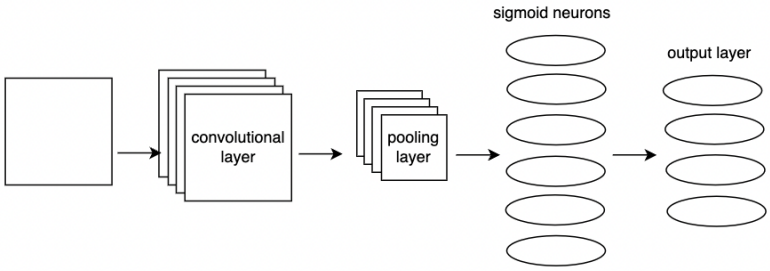

Figure 3.6. Convolutional Neural Networks, Adapted from [50].

Figure 3.6 shows the CNN process. The pooling layer here comes differently. This layer simplifies what the convolutional layer does, takes the results of convolutional layers and processes them [50].

3.5.3. Recurrent Neural Networks (RNN)

Strength of Recurrent Neural Networks depends on the ability to remember previous data which is done in hidden layers [56].

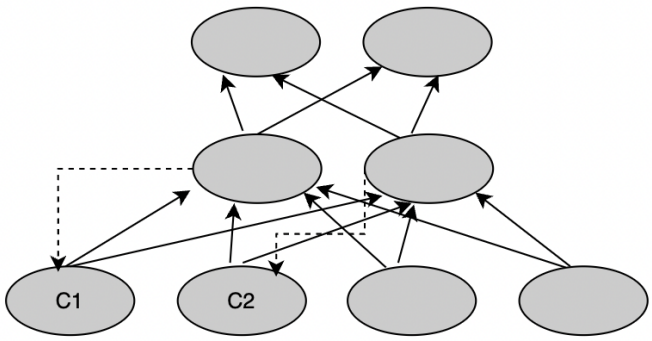

Figure 3.7. Recurrent Neural Networks, Adapted from [54].

In Figure 3.7, *C1* and *C2* letters are used to represent context units. While processing the previous state, RNN gives different weights and does not process all the data again directly. It focuses on the most weighted features [56].

RNN is mostly used for sequential data. Tasks like handwriting and speech recognition are also processed by this [57]. The advantage of RNN is that it can process huge numbers of data as well, but its model does not increase much according to the size of the input. However, it is slow for computation and cannot reach much earlier data [57].

3.5.4. Bidirectional Recurrent Neural Networks

RNN networks can process past data and conclude accordingly, but for some predictions such as speech recognition, it is necessary to take upcoming words into account for a meaningful result. That is why Bidirectional Recurrent Neural Networks work with two RNNs to process the input from right to left and left to right together [58].

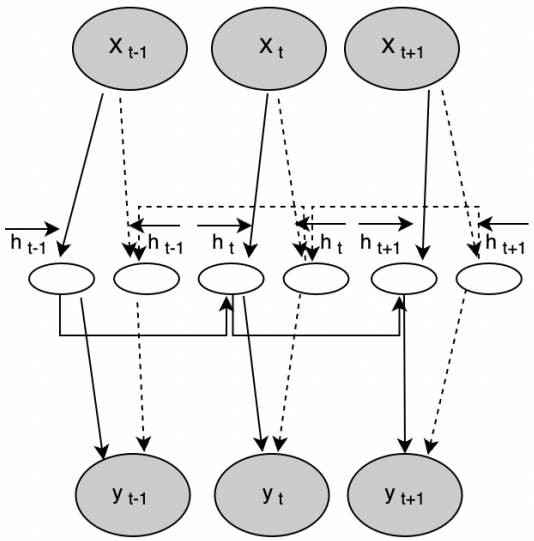

Figure 3.8. Bidirectional Recurrent Neural Networks, Adapted from [58].

In Figure 3.8, x and y letters are used to represent inputs, while h letter is used to represent output which is given with t as related time sequence [58].

With two hidden states, Bidirectional RNNs combine forward and backward processes, but the dimension can be increased to four with up and down processes as well [52].

3.5.5. Long Short Term Memory Neural Networks (LSTM)

Long Short Term Memory (LSTM) algorithms are a branch of RNN algorithms that can relate dependencies between sequential data like words in sentences. While relating earlier parts of sequences is hard in RNN, LSTM brings a solution to this problem which is called the vanishing gradient problem solution [58].

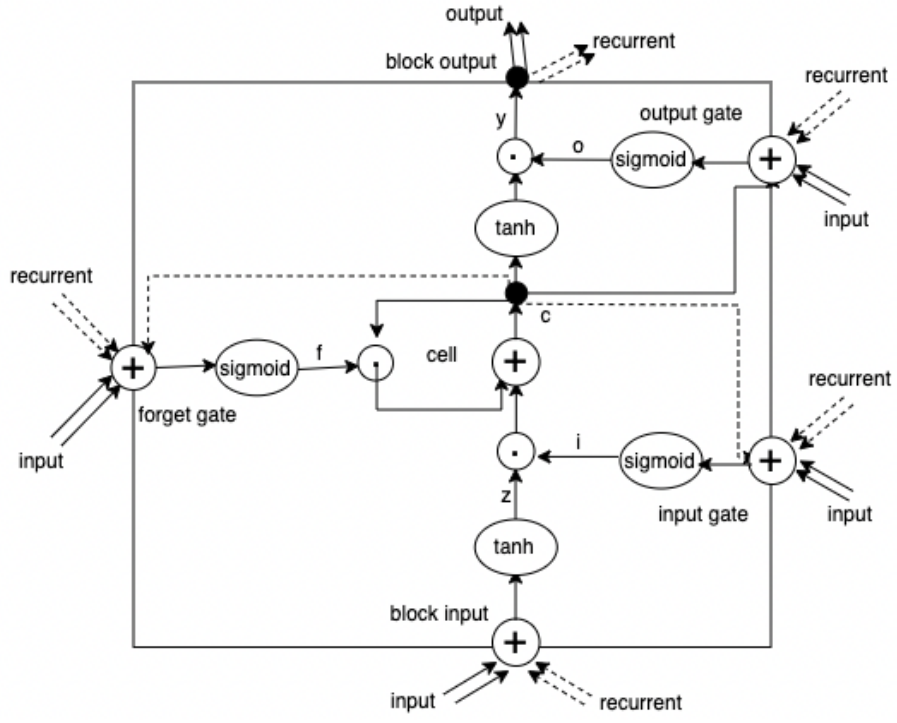

Figure 3.9. Long Short Term Memory Cell, Adapted from [59].

Thanks to the forget gate, LSTM deletes unnecessary information after processing it with the activation function. This algorithm is useful for unlimited time sequences with input, output, and forget gates [60]. LSTM architecture and usage are explained with details in Chapter 5.

3.5.6. Gated Recurrent Unit (GRU)

After the discussion of which part of LSTM is needed for some cases, Gated Recurrent Unit comes to the scene. It differs from LSTM by having a single gate unit that checks forgetting and decision at the same time [52].

According to the hidden memory state and current input, the reset gate and update gate are calculated and the new hidden state is decided accordingly [58].

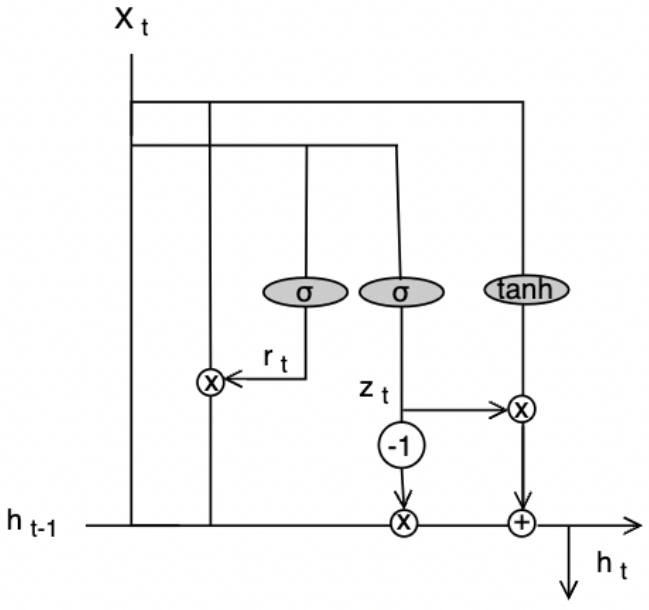

Figure 3.10. Gated Recurrent Unit Architecture, Adapted from [58].

In Figure 3.10, x is the current input while h is the hidden state output. r and z with related time represent reset and update gates. σ and $tanh$ are the functions used to calculate the hidden state output [58].

3.6. Federated Learning

Federated learning means a machine learning or deep learning environment with multiple collaborators, also referred as clients, that are coordinated under a central service provider. There is a huge data set in clients' local machines because the sensors collect the data easily and repeatedly. These collaborators' data is collected in their local environment and not transmitted to the central area. The data from clients are used to update the learning of the central model [61]. Federated learning has advantages like separating the data and achieving privacy security. There is a need to train the central model so that the available data from each client can be used to update it without security risks [62].

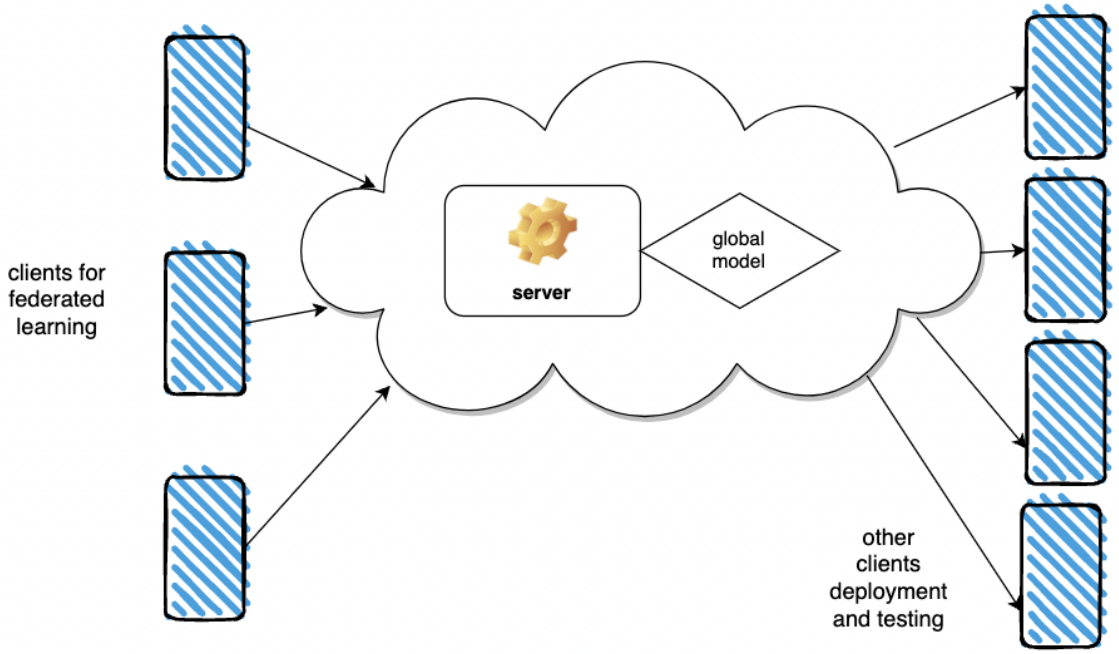

Figure 3.11. Federated Learning Architecture, Adapted from [61].

Other than device-based federated learning, cross-silo federated learning is also mentioned in the literature. With decentralized local data collected from different organizations in the areas like finance or medicine, hubs in the topology are connected to the central, and there might be hundreds of users for this type of architecture [61].

The process of federated learning includes steps such as defining the problem, client instrumentation, prototyping as simulation, training federated model, and deployment [61].

4. MATERIALS AND METHODS

4.1. Collected Data

We collected the data we use in the experiment from public transportation passengers in Istanbul via the Android application that we developed. To implement our study with the same type of vehicle, we only included the buses in our data collection part. Because it is important to have the same type of path in the experiment to not be affected by differences on the road, we included the path between Harbiye - Unkapanı and Unkapanı - Harbiye stations in Istanbul, Turkey. We predicted it to take approximately 15 minutes to go. The passengers recorded the two paths in different sessions. We got 64 session records for each direction which is a totally 128 number of session records.

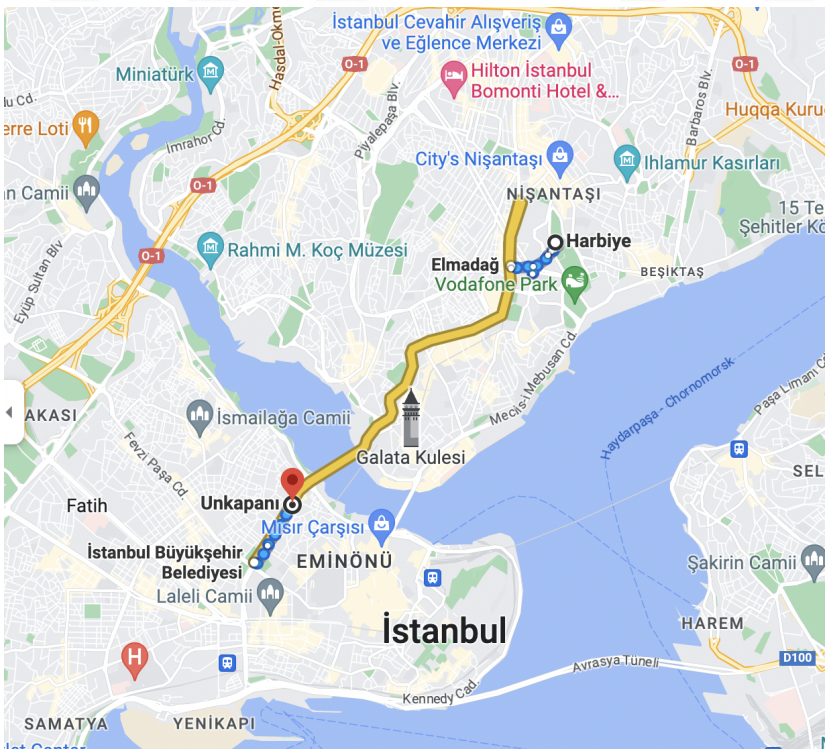

Figure 4.1. Harbiye - Unkapanı Path.

In Figure 4.1, the bus path selected with a yellow line is one of the directions we mentioned above.

Passengers downloaded our application before joining the experiment and gave permission for the application to collect their smartphone sensor data. When they got on the bus, they started the session and used their phones only in the standard forward horizontal direction. As the journey continued until the passengers finished it, the session recorded the accelerometer and gyroscope sensor data shown on the screen simultaneously. Passengers ended the session just before getting off the bus and voted on the driving behavior. The options on the voting page were listed as normal, aggressive, or risky. The sessions were never started or finished in any other location than the stations we defined earlier even if the journey would take shorter or longer than we estimated. The mobile phones that we collected the data on were of three models: Poco X3, Poco X3 Pro, and Samsung A53. The sensors in the phones were named lsm6dso Non-wakeup, bmi220 Non-wakeup and LSM6DSO.

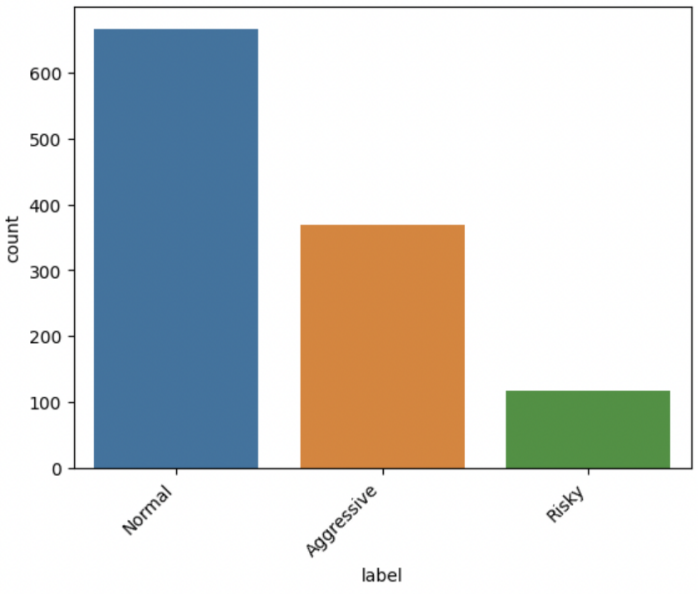

Figure 4.2. Data Label Distribution with 3 Label Options.

The voting results distribution for the total number of sessions in two directions of the path is shown in Figure 4.2.

To have a larger data set, we used both paths of Harbiye - Unkapanı and Unkapanı - Harbiye separately and combined them in different experiments. We implemented experiments with the data voted with 3 labels and processed the data to turn into 2 labels. Because detecting risky driving behavior is important, we divided the votes into two categories as safe and danger where danger only includes voted sessions with risky labels.

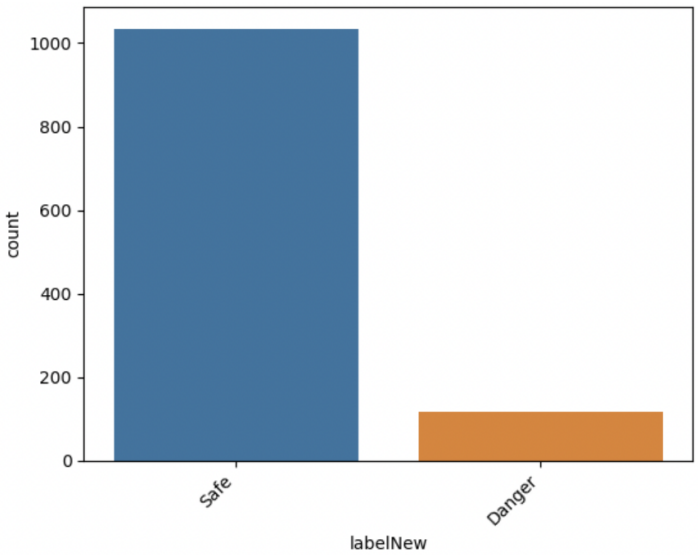

Figure 4.3. Data Label Distribution with 2 Label Options.

The processed data distribution with 2 labels for the total number of sessions of the two directions are shown in Figure 4.3.

The passengers shared their session records on their smartphones with us via a cloud environment.

4.2. Mobile Application

We developed an Android application to collect the data from passengers for detecting the changes in their accelerometer and gyroscope data during the session. On Android phones, there are several sensors such as accelerometer, accelerometer uncali-

brated, gravity, gyroscope, gyroscope uncalibrated, linear acceleration, rotation vector, significant motion, step counter, and step detector, which can directly be reached via an Android application code [63]. We only used accelerometer and gyroscope types of sensors in our application and study.

There are three main activities in the application: starting a session, collecting sensor data, and ending the session to vote on the driving behavior.

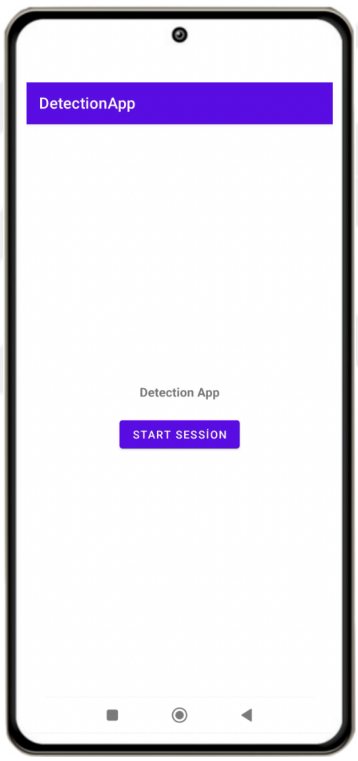

Figure 4.4. Mobile Application Session Start Screen.

When the button to start the session was clicked on the start page, our mobile application started to record the smartphone's accelerometer and gyroscope sensors' x, y, and z axis data with 1 Hz frequency to the separate CSV files that it created under the session folder within the application folder in the downloads sub-folder of the smartphone.

To end the session, users pressed on the end session button on the sensor page of the application. When the session ended, passengers voted the behavior of the driver of that session as normal, aggressive, or risky from the dropdown list on the voting page. When users pressed on the record voting button, our application recorded the vote on a TXT file under the session folder and showed the feedback to the user as the given feedback was recorded.

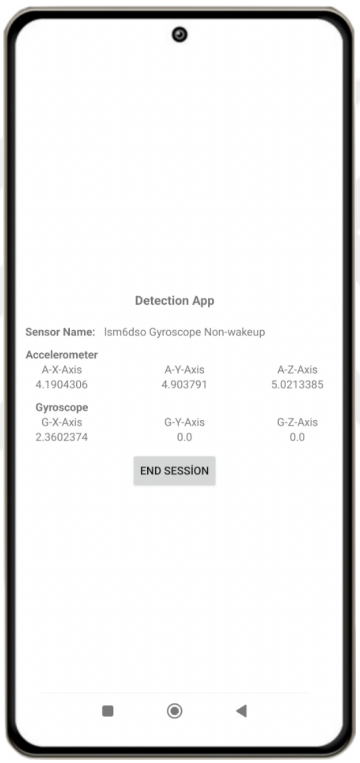

Figure 4.5. Mobile Application Sensor Data Screen.

To start another session, passengers restarted the application. Then, when they pushed the start session button, a new session folder was created under the application folder in the downloads sub-folder of the smartphone.

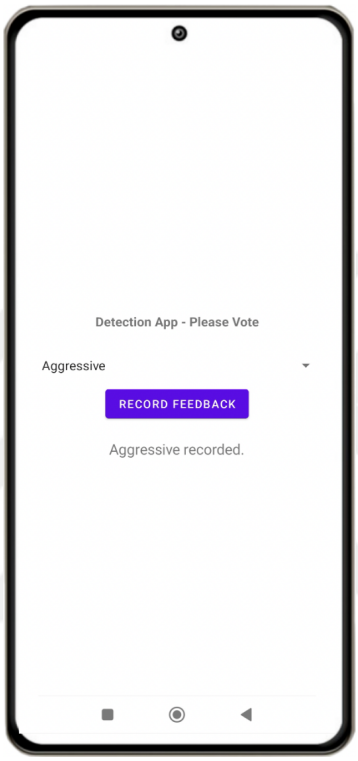

Figure 4.6. Mobile Application Voting Screen.

Because the data is recorded on the smartphone storage, we wanted passengers to share their data with us via a cloud environment. We got 64 useful sessions for each direction which is 128 sessions in total.

Each session had its folder and those folders included accelerometer, gyroscope, and vote files separately. The name of the session folder was the timestamp of the starting time and voting was also recorded with the timestamp of the voting time.

We gathered the session folders from the users and kept them in their related timestamp folders. Even if we processed the data to use in our experiments, we kept the original data separately.

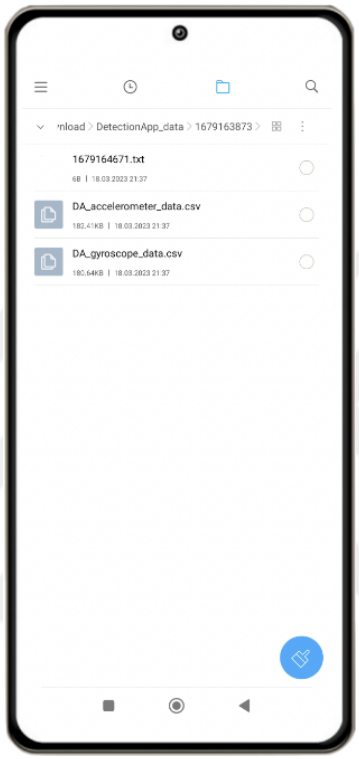

Figure 4.7. Mobile Application Session Folder Content.

The content of a particular session folder can be seen in Figure 4.7 which includes accelerometer, gyroscope, and vote files.

Our mobile application sequence diagram also includes the process of the session and activity codes.

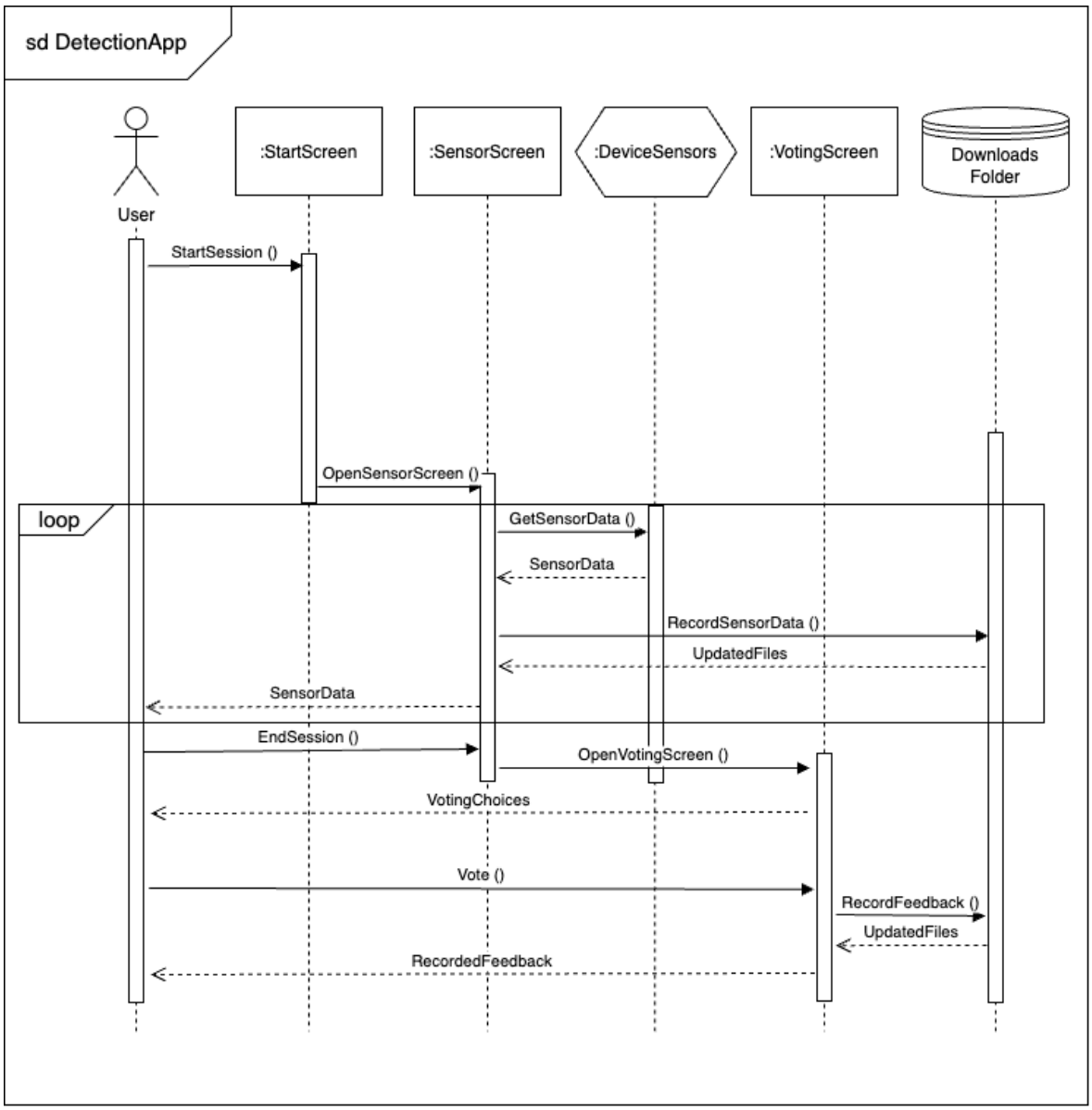

Figure 4.8. Mobile Application Sequence Diagram.

In Figure 4.8, the sensor data collection part represents a loop because the data was collected during the whole session time. Other than sensor data, other activities were triggered by the user.

4.3. Data Processing

To have a meaningful data set with useful features, at first we windowed the sensor data as one minute for each row. For this purpose, we used the resampling method by applying some functions [64]. For each session data with accelerometer and gyroscope, after applying our functions, we got the same number of rows with how many minutes any session takes. Because we applied different experiment methods, while keeping this number of minute rows, we took 9 minutes of data, so 9 rows of data in order to have the same number of rows for all sessions because we processed it with a deep learning algorithm for time series in the experiment. We decided on 9 rows because the shortest session took 9 minutes.

We also processed the raw data with the same functions to have 10-second rows for each session. We applied this to process the featured data to get minute-wise predictions later in the experiment.

For the voting result of each session, we created a new column both for 1-minute and 10-second types of resampled data and entered the vote there in each row. This column represented a label for us, and it was an actual value to compare with the predicted one. We also created another column to write the session number of the session but did not use this as a feature.

The functions used to process the data can be listed as follows:

Mean: The mean function of the Python math library was applied to x, y, and z-axis of the sensor to resample as minutes and seconds. Results were recorded as the first letter of the sensor name, accelerometer (A) or gyroscope (G), with the axis name and the abbreviation of the function name which were AX_mean, AY_mean, AZ_mean, GX_mean, GY_mean, and GZ_mean. Then, the resulting axis mean was used when all x, y, and z directions' data was processed together for a sensor-based interpretation. One of the processes was getting the mean of the axes' mean values. The results were written as the abbreviation of the function name with the first letter of the sensor name and XYZ that represents all axes which were mean_AXYZ and mean_GXYZ.

Standard Deviation: The standard deviation function of the Python math library was applied to x, y, and z-axis to resample as minutes and seconds. Results were recorded as AX_std, AY_std, AZ_std, GX_std, GY_std and GZ_std. Then, it was also applied to the mean value of sensor axes to find the standard deviation which was recorded as std_AXYZ and std_GXYZ.

Variance: We applied the variance function of the Python math library to x, y, and z-axis to resample as minutes and seconds. We recorded the results as AX_var, AY_var, AZ_var, GX_var, GY_var and GZ_var. Then, it was applied to the mean value of sensor axes to find their variance recorded as var_AXYZ and var_GXYZ.

Minimum: The minimum function of the Python math library was applied to the sensor's x, y, and z-axis to resample as minutes and seconds. Results were recorded as AX_min, AY_min, AZ_min, GX_min, GY_min and GZ_min. Then, it was applied to the mean value of sensor axes to find the minimum recorded as min_AXYZ and min_GXYZ.

Maximum: The maximum function of the Python math library was also applied to the sensor's x, y, and z-axis to resample as minutes and seconds. Results were recorded as AX_max, AY_max, AZ_max, GX_max, GY_max and GZ_max. Then, it was applied to the mean value of sensor axes to find their maximum recorded as max_AXYZ and max_GXYZ.

Median: The median function of the Python math library was only applied to the mean value of sensor axes to find their median which were recorded as median_AXYZ and median_GXYZ.

Magnitude: The magnitude of the mean values of axes was expressed as

$$\sqrt{mean_x^2 + mean_y^2 + mean_z^2} \tag{4.1}$$

where $mean_x$, $mean_y$ and $mean_z$ represents mean value of the x, y and z axes. Results were recorded as magnitude_AXYZ and magnitude_GXYZ.

Signal Magnitude Area: The received signal was expressed as

$$\frac{\sqrt{mean_x^2 + mean_y^2 + mean_z^2}}{3} \tag{4.2}$$

where $mean_x$, $mean_y$ and $mean_z$ represents mean value of the x, y and z axes. Results were recorded as signalMagnitudeArea_AXYZ and signalMagnitudeArea_GXYZ.

Root Mean Squared: The received signal was expressed as

$$\sqrt{\frac{mean_x^2 + mean_y^2 + mean_z^2}{3}} \tag{4.3}$$

where $mean_x$, $mean_y$ and $mean_z$ represents mean value of the x, y and z axes. Results were recorded as rootMeanSquared_AXYZ and rootMeanSquared_GXYZ.

Curve Length: The received signal was expressed as

$$|mean_x - mean_y| + |mean_y - mean_z| \tag{4.4}$$

where $mean_x$, $mean_y$ and $mean_z$ represents mean value of the x, y and z axes. Results were recorded as curveLength_AXYZ and curveLength_GXYZ.

Time	AX_mean	AX_std	AX_var	AX_min	AX_max	AY_mean	AY_std	AY_var	AY_min	AY_max	...	max_AXYZ	min_AXYZ	std_AXYZ	median_AXYZ	magnitude_AXYZ	signalMagnitudeArea_AXYZ	rootMeanSquared_AXYZ	curveLength_AXYZ
2023-03-18 10:58:00-03:00	0.050033	0.373553	0.139542	0.0	3.946662	0.000000	0.000000	0.000000	0.0	0.000000	...	0.121321	0.000000	0.060970	0.050033	0.131233	0.043744	0.075767	0.171354
2023-03-18 10:59:00-03:00	0.000000	0.000000	0.000000	0.0	0.000000	0.024905	0.253233	0.064127	0.0	2.877369	...	0.044507	0.000000	0.022306	0.024905	0.051002	0.017001	0.029446	0.044507
2023-03-18 11:00:00-03:00	0.000000	0.000000	0.000000	0.0	0.000000	0.021706	0.272366	0.074183	0.0	3.798905	...	0.021706	0.000000	0.012532	0.000000	0.021706	0.007235	0.012532	0.043411
2023-03-18 11:01:00-03:00	0.000000	0.000000	0.000000	0.0	0.000000	0.007615	0.134069	0.017974	0.0	2.360520	...	0.007615	0.000000	0.004396	0.000000	0.007615	0.002538	0.004396	0.015229
2023-03-18 11:02:00-03:00	0.013913	0.172974	0.029920	0.0	2.201696	0.006460	0.113734	0.012935	0.0	2.002493	...	0.013913	0.006460	0.004135	0.007082	0.016895	0.005632	0.009755	0.008076
2023-03-18 11:03:00-03:00	0.065631	0.448527	0.201177	0.0	4.527819	0.083685	0.472896	0.223631	0.0	4.155436	...	0.154457	0.065631	0.046948	0.083685	0.187530	0.062510	0.108270	0.088826
2023-03-18 11:04:00-03:00	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	...	0.017582	0.000000	0.010151	0.000000	0.017582	0.005861	0.010151	0.017582
2023-03-18 11:05:00-03:00	0.022260	0.227106	0.051577	0.0	2.677269	0.000000	0.000000	0.000000	0.0	0.000000	...	0.022260	0.000000	0.012852	0.000000	0.022260	0.007420	0.012852	0.022260
2023-03-18 11:06:00-03:00	0.028479	0.292969	0.085831	0.0	3.366103	0.000000	0.000000	0.000000	0.0	0.000000	...	0.028479	0.000000	0.016443	0.000000	0.028479	0.009493	0.016443	0.028479
2023-03-18 11:07:00-03:00	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2023-03-18 11:08:00-03:00	0.056444	0.413034	0.170597	0.0	4.287938	0.056974	0.386921	0.149708	0.0	3.904189	...	0.056974	0.035627	0.012175	0.056444	0.087757	0.029252	0.050666	0.021878
2023-03-18 11:09:00-03:00	0.007557	0.132628	0.017590	0.0	2.327618	0.028224	0.299141	0.089485	0.0	4.195216	...	0.028224	0.007557	0.011772	0.008125	0.030327	0.010109	0.017510	0.040766
2023-03-18 11:10:00-03:00	0.097126	0.576593	0.332460	0.0	6.384350	0.040359	0.355052	0.126062	0.0	3.366103	...	0.132243	0.040359	0.046365	0.097126	0.168969	0.056323	0.097554	0.148652

Figure 4.9. Accelerometer Resampled Data Example.

After applying the functions, we got 50 features to include in the experiment. The first 25 were for the accelerometer and the other 25 were for the gyroscope. Of the 25 features for each sensor, the first 15 features were the results for the x, y, and z axis where 5 columns for each axis was calculated with mean, standard deviation, variance, minimum and maximum functions. The other 10 columns for each sensor were the result of variance, mean, maximum, minimum, standard deviation, median, magnitude, signal magnitude area, root mean squared, and curve length functions which were applied to the mean values of the sensor axes. We collected each path's session results in different files and folders.

5. TECHNICAL DETAILS

As described earlier, we collected the data from passengers' smartphones and processed them with functions as 1 minute and 10 seconds wise. Because we wanted to detect the real dangerous driving behaviors, we implemented the experiments on the data with 3-label votes and 2-label votes separately. To detect a dangerous situation, we relabeled "risky" as "danger", and "normal" and "aggressive" together as "safe". We used both machine learning and deep learning algorithms to predict the driving behavior.

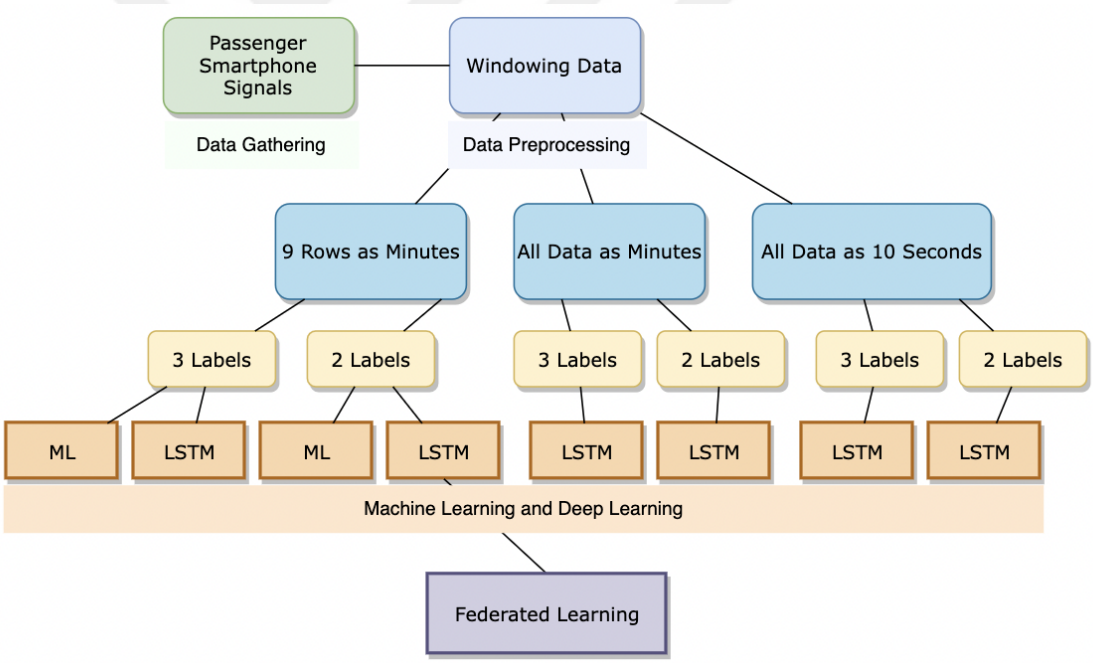

Figure 5.1. Process of the Experiment.

To see the results of machine learning, we used 9-minute windows of each session as input as the combination of two directions and concluded minute-wise predictions. For the LSTM algorithm, we re-used 9-minute windowed sessions to have sequence division with sessions but also generated inputs with all minutes of sessions with a probabilistic approach and 10-second-wise inputs to predict each minute behavior.

5.1. Machine Learning Classification

Firstly, we used machine learning algorithms to predict driving behaviors. We implemented the most common machine learning algorithms with Python scikit-learn library default parameters.

We divided our data into two as 0.7 train and 0.3 test. We did not randomize it while dividing because having the minutes in order was important for session-based interpretations. We dropped the time and session columns because they were not used in the experiment. Only 50 features mentioned previously were used as input with 1 label for driving behavior. After applying the standard scaling function of the pre-processing ability of the library, we implemented Logistic Regression, Support Vector Machines, K Neighbors Classifier, Decision Tree Classifier, Random Forest Classifier, and Gaussian Naive Bayes algorithms.

5.2. Long Short Term Memory Algorithm Architecture

To develop our Long Short Term Memory (LSTM) algorithm, we used the Python Pytorch library. We dropped time and session columns as they were not included in features and selected 50 features as input with 1 label. Firstly, because we had 50 features in the data input with resulted number of rows of data, there were two dimensions. Therefore, we extended label input to two dimensions as well with row numbers. The X and y inputs were like (number of rows, 50) and (number of rows, 1) respectively with 50 features and 1 label. Then, we applied the standard scalar function of the Python scikit-learn library to normalize the feature data. We preferred Standardization, not Normalization because we wanted to scale the data with mean by dividing standard deviation to shift its distribution so as to get a mean of zero and standard deviation of one instead of scaling it to range 0-1 for floating-point values [65]. We also applied Label Encoder in order to turn the label text data into numerical classification.

For session driving behavior prediction, we reshaped our data with 9 because we had 9 minutes for each session. When we were using it for minute prediction, we reshaped it with 6. For 9 minutes of the session, we should have 50x9 two-dimensional data.

To define labels for series, we defined a prediction matrix and calculated how many of which labels were there in that shape.

We divided our data set into two as 0.7 for the train and 0.3 for the test, but did not randomize it. We converted our set to a torch. Then, we applied the softmax function to get probability numbers.

Softmax Function: This function is applied to get the corresponding weights of numbers as a probability. When the counts of inputs are given, it gets the probability.

$$a_j^L = \frac{e^{x}{}_j^L}{\sum_k e^{z}{}_k^L} \tag{5.1}$$

where z numbers are the probable numbers and a is the corresponding output [50].

To prepare our data set for the loader for giving into the LSTM algorithm, we turned the shape into two dimensions again which have the number of sessions in the train or test set and the number of labels.

We used some parameters for the LSTM algorithm and defined the algorithm with them. These parameters are input size, sequence length, number of layers, hidden size, number of classes, learning rate, batch size, and number of epochs differed for the experiments. Input size means the number of features we give as input, which is 50 for us in each experiment. Sequence length is the number of lengths for each session. When we take a 9-minute session, this parameter should be 9. Hidden size gives the weighted features in connected layers in the hidden state. After adding all, it also tries to add bias value and gets the best learning.

The parameter referred as number of classes is the label counts we want to predict. Batch size is the number of parts to be taken at each time in the algorithm. Epoch is how many times this process is repeated [52, 58, 66].

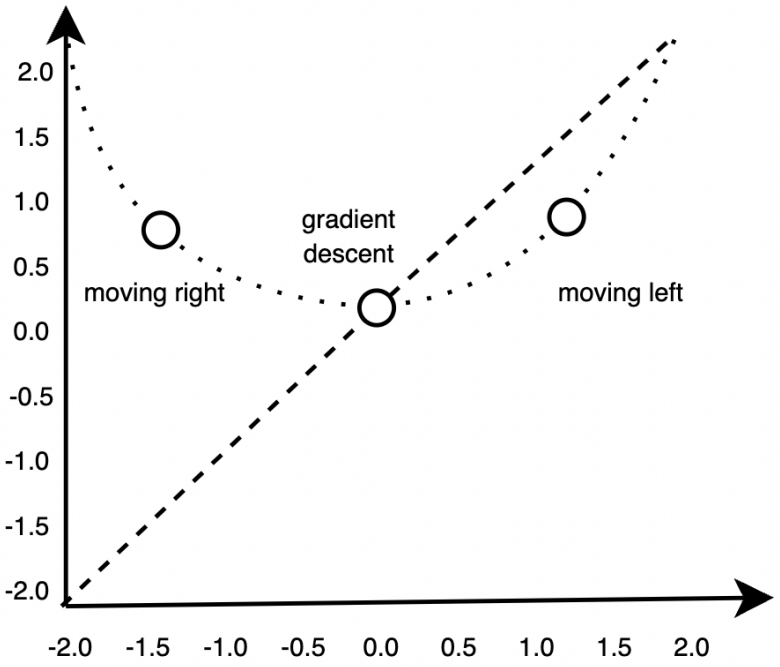

Figure 5.2. Gradient Descent Structure, Adapted from [52].

While predicting the result, the algorithm compares the values. Hence, by taking the derivative, it continues learning by back-propagation [58]. Here, the learning rate affects how long it will move. When the learning rate is too low, catching the exact value takes much time to move, which can be seen in Figure 5.2. If it is too high, the move might pass the point it is supposed to be.

Long Short Term Memory (LSTM) algorithm is derived from a neural network module that takes each sample of sequence. The input, output, and forget gates are used inside. There are also activation functions used inside the LSTM architecture.

Sigmoid Function: This function is preferred for deep learning algorithms as an activation function. It takes the real value and reduces it to the range (0,1).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.2}$$

There are other activation functions such as tanh and rectified linear unit (ReLU), but sigmoid is chosen for outputs in range (0,1) [67].

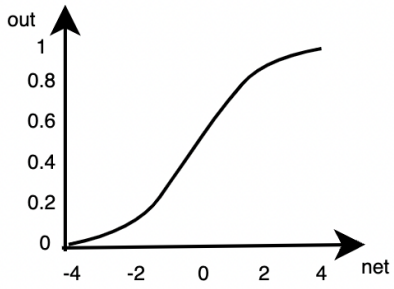

Figure 5.3. Sigmoid Function, Adapted from [67].

Tanh Function: This function is also used in the LSTM algorithm as an activation function. It is the scaled version of the sigmoid function

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{5.3}$$

which is also used in LSTM architecture [67].

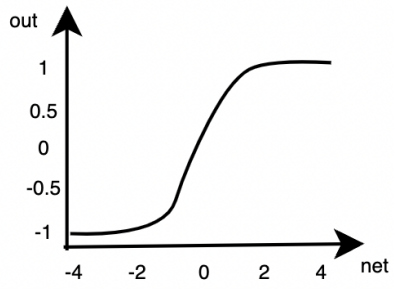

Figure 5.4. tanh Function, Adapted from [67].

In LSTM algorithm architecture, f_t is used for the "do not forget gate" while i_t and o_t are input and output gates. The functions used to calculate the results are listed below [66, 68].

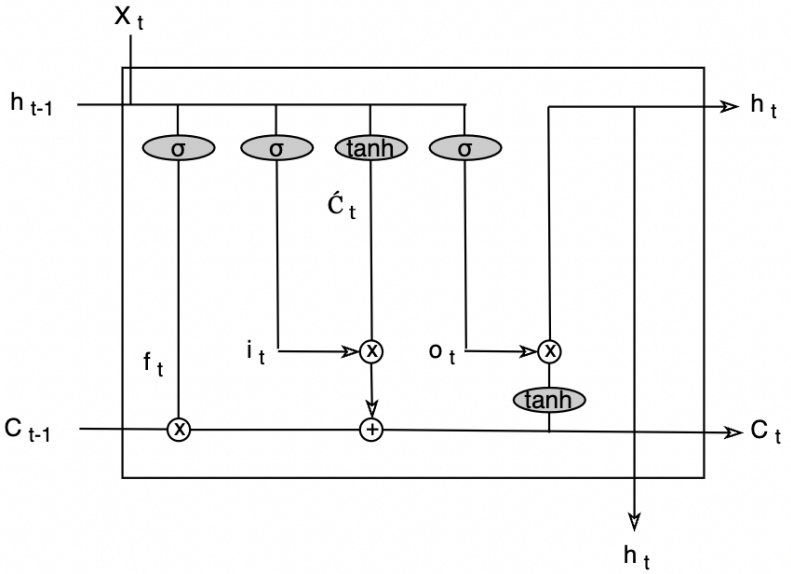

Figure 5.5. Long Short Term Memory Algorithm Architecture, Adapted from [66].

In time t-1, input enters the algorithm and outs in time t. C represents long-term memory, cell state, and h represents short-term memory. At first, the input is processed by the sigmoid (σ) function.

Forget Gate: This function gets a weighted summation of the cell in time t, the output of the cell in time *t-1*, and the input in time t to calculate the output of the forget gate with the activation function. It decides if the coming part is related and should be remembered or not. If it is about 0 when calculated from the activation function, that means it is more likely to be forgotten.

$$f_t = \sigma(W_f \ . \ [h_{t-1}, x_t] + b_f) \tag{5.4}$$

where the activation function is sigmoid. Here, W_f represents the weight, b_f is the bias vector, h_{t-1} is the output of the previous time and x_t is the input at that time.

Input Gate: This function gets weighted summation again. Here, short-term memory affects the long-term memory which is used to update the cell.

$$i_t = \sigma(W_i \ . \ [h_{t-1}, x_t] + b_i) \tag{5.5}$$

where the activation function is sigmoid. Here, W_i represents the weight, b_i is the bias vector, h_{t-1} is the output of the previous time and x_t is the input at that time.

Candidate Gate: This function yields the result of candidate value for the memory unit at that time. It calculates the value with the tanh activation function.

$$\widetilde{C}_t = tanh(W_C \ . \ [h_{t-1}, x_t] + b_c) \tag{5.6}$$

where W_C represents the weight, b_c is bias vector, h_{t-1} is the output of previous time and x_t is the input at that time.

Cell: This function brings the memories together for the past and present.

$$C_t = f_t \ * C_{t-1} + i_t \ * \widetilde{C}_t \tag{5.7}$$

where C_t represents the cell, f_t is forget gate, C_{t-1} is previous cell, i_t is input gate and $\widetilde{C}_t$ is candidate gate.

Output Gate: This function gives the result of weighted summation again.

$$o_t = \sigma(W_o \ . \ [h_{t-1}, x_t] + b_o) \tag{5.8}$$

where the activation function is sigmoid. Here, W_o represents the weight, b_o is the bias vector, h_{t-1} is the output of the previous time and x_t is the input at that time.

Hyperbolic Tangent Version of The Cell: This function gives the cell output. The activation function is tanh here.

$$h_t = o_t * tanh(C_t) \tag{5.9}$$

where o_t is output gate and C_t is the cell.

As an optimizer, we chose Adam optimizer for our LSTM algorithm. There are AdaGrad, RMSprop, Adaptive Moment Estimation (Adam), and AdaDelta optimizers for adaptive learning [52]. AdaGrap optimizer synchronizes learning rates according to model parameters. Parameters yielding the result of large derivatives have a huge decrease in the learning rate in this optimizer and weight cannot be calculated when the learning rate is too low. RMSprop optimizer changes gradient accumulation to weighted average in order to move accordingly, which is very similar to AdaDelta. AdaDelta algorithm directly searches for the address. It solves the vanishing gradient problem with a history list. Adam optimizer comes with more advantages when compared with AdaDelta and RMSprop. Even if the optimizer selection depends on the algorithm to be implemented, the Adam optimizer is generally found the best algorithm by many users. Learning rate does not get so much importance here, and it works well with sparse data as well. This optimizer applies exponential weighting and has bias corrections [52, 69].

After applying the LSTM algorithm, we again applied the softmax function to the output so as to have the probabilistic result. We compared the predicted and actual values and thus calculated accuracy.

Loss results are the losses from each time step when the algorithm is applied [55]. We drew a loss graph after implementing the LSTM algorithm.

5.3. Federated Learning Architecture

Having a global model to refer to different devices as well as data privacy are important in data analysis technologies. That is why we implemented a federated learning algorithm with our LSTM algorithm.

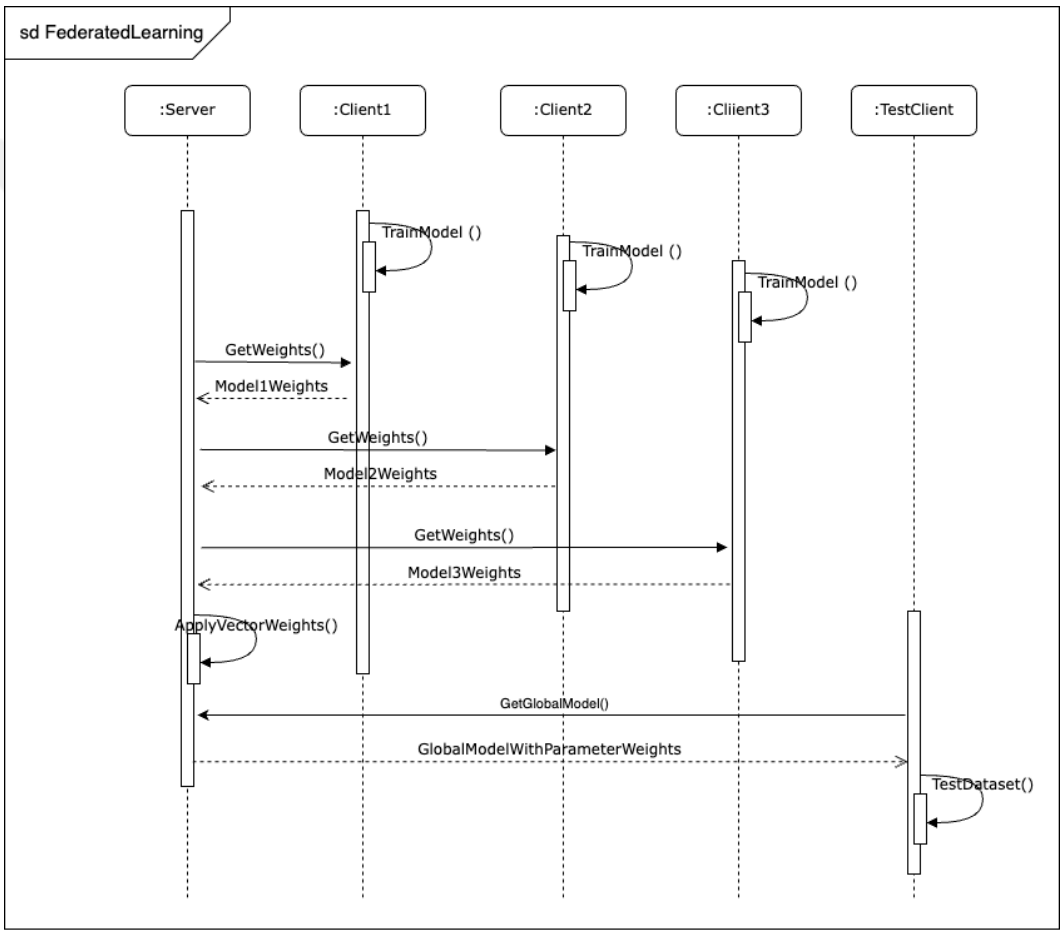

Figure 5.6. Federated Learning Sequence Diagram.

We divided our data set into two as 0.7 train and 0.3 test. For the train data, we divided it into three to have 3 different client models. We created 3 different models for each data set and applied our LSTM algorithm separately. After getting the trained models for each, we turned the model parameters into vectors and summed these vectors to assign the result to the global model. After this process, we were able to test our test data set with a global model.

6. EXPERIMENTAL RESULTS

6.1. Machine Learning Classification

6.1.1. 3 Labels Experiment with 9 Minutes Session

To implement machine learning algorithms, we generated minute-wise windowed data set from our sensor data collected from passengers. We took 9 minutes of each session and combined the two path results which were Harbiye - Unkapanı and Unkapanı - Harbiye.

Table 6.1. Machine learning algorithms accuracy list for 3 labels.

Algorithm Name	**Accuracy**
Logistic Regression	54.62 %
Support Vector Machine	53.75 %
K-Nearest Neighbors	50.00 %
Decision Tree	45.95 %
Random Forest	52.60 %
Naive Bayes	55.20 %

Table 6.1 lists the accuracy results of the input data with 3-label classifications as normal, aggressive and risky.

6.1.2. 2 Labels Experiment with 9 Minutes Session

To find dangerous driving behaviors, we relabeled the data as "safe" and "danger". For this data, we used minute-wise windowed data set from our sensor data from passengers again. We took 9 minutes of each session and combined the results of two

paths which were Harbiye - Unkapanı and Unkapanı - Harbiye.

Table 6.2. Machine learning algorithms accuracy list for 2 labels.

Algorithm Name	**Accuracy**
Logistic Regression	93.06 %
Support Vector Machine	92.19 %
K-Nearest Neighbors	89.88 %
Decision Tree	82.36 %
Random Forest	92.19 %
Naive Bayes	92.19 %

Table 6.2 lists the accuracy results of the input data with 3-label classifications as normal, aggressive and risky.

6.2. Long Short Term Memory Algorithm

6.2.1. 3-Label Experiment with All Minutes

The main aim of our study is to increase accuracy. Therefore, we implemented our experiments with the LSTM algorithm as well because it is suitable to use with time series data. We used the parameters listed below for the LSTM algorithm and implemented it on all sessions with all minute rows of Harbiye - Unkapanı data.

- Input Size: 50
- Sequence Length: 9
- Number of Layers: 2
- Hidden Size: 256
- Number of Classes: 3
- Learning Rate: 0.00001
- Batch Size: 10
- Number of Epochs: 50

With this all-minutes data set which cannot be divided by sessions as sequences with the same amount of minutes, we got 72.73 % accuracy on the train and 61.76 % on the test. We can see the loss graph below.

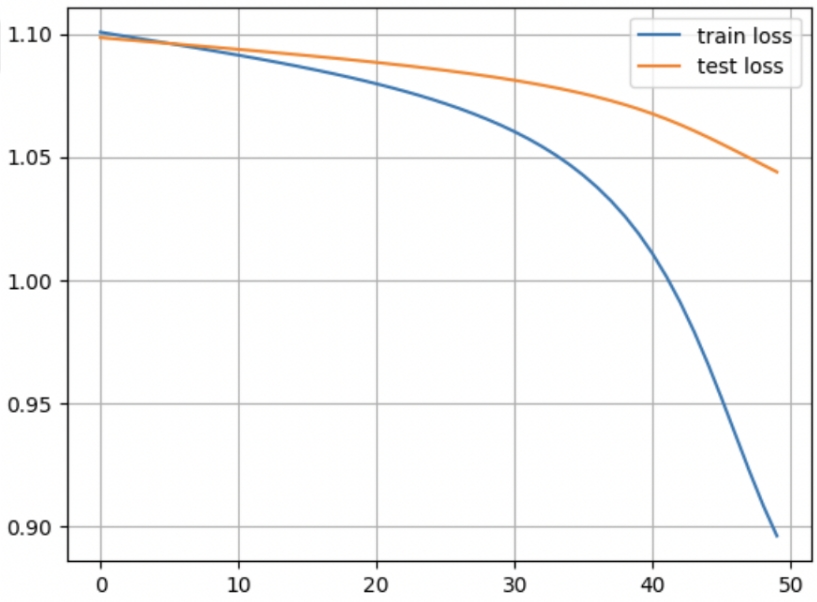

Figure 6.1. Long Short Term Memory Algorithm Loss Graph for 3 Labels with All-Minutes Data.

6.2.2. 2-Label Experiment with All Minutes

Our main focus is to detect real dangerous driving behavior. Therefore, we re-labeled the votes as "safe" and "danger" which was explained in Chapter 4. We used

the parameters listed below for the LSTM algorithm and implemented it on the all sessions with all minute rows of Harbiye - Unkapanı data.

- Input Size: 50
- Sequence Length: 9
- Number of Layers: 2
- Hidden Size: 256
- Number of Classes: 2
- Learning Rate: 0.0001
- Batch Size: 10
- Number of Epochs: 10

With this all-minutes data set which cannot be divided by sessions as sequences with the same amount of minutes, we reached 90.91 % accuracy on the train and 88.24 % on the test. We can see the loss graph below.

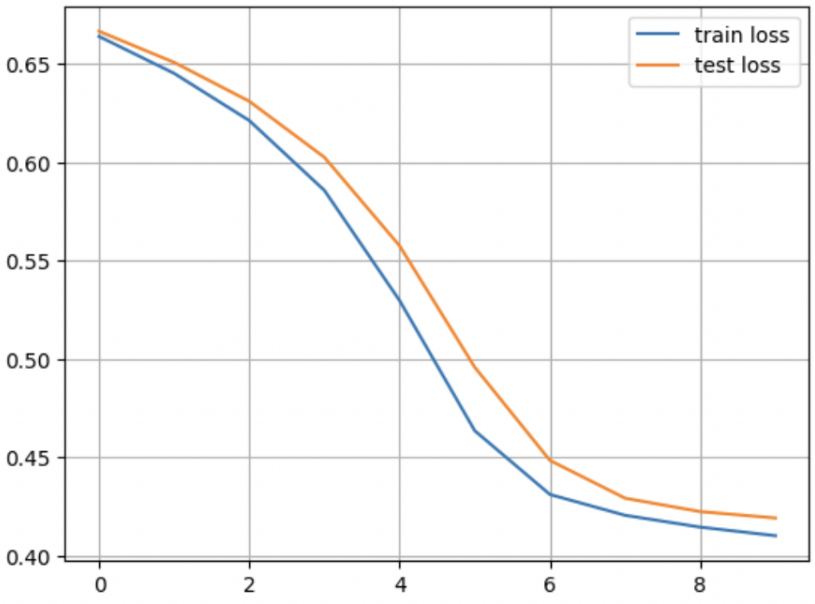

Figure 6.2. Long Short Term Memory Algorithm Loss Graph for 2 Labels with All-Minutes Data.

6.2.3. 3-Label Experiment with 10-Second Sessions to Predict Minute Label

To experiment with how our LSTM algorithm predicts minute-wise labels, we generated 10 seconds of windowed data from Harbiye - Unkapanı direction. We used the parameters listed below for the LSTM algorithm and implemented it on all sessions with all seconds rows of Harbiye - Unkapanı data. We took the sequence as 6 to be able to divide each minute which is 60 seconds as sequences.

- Input Size: 50
- Sequence Length: 6
- Number of Layers: 2
- Hidden Size: 256
- Number of Classes: 3
- Learning Rate: 0.001
- Batch Size: 1000
- Number of Epochs: 20

With this all-seconds-included set of data that cannot be divided by sessions as sequences, we got 68.51 % accuracy on the train and 56.31 % on the test. We compared predicted and actual values for each session because we were able to see all minute-wise predictions for all sessions, we knew how many minutes each session took, and the data was not randomized. We calculated the accuracy for each session and got the mean value of them. In the end, we were able to get 64.90 % accuracy.

The loss graph is given below.

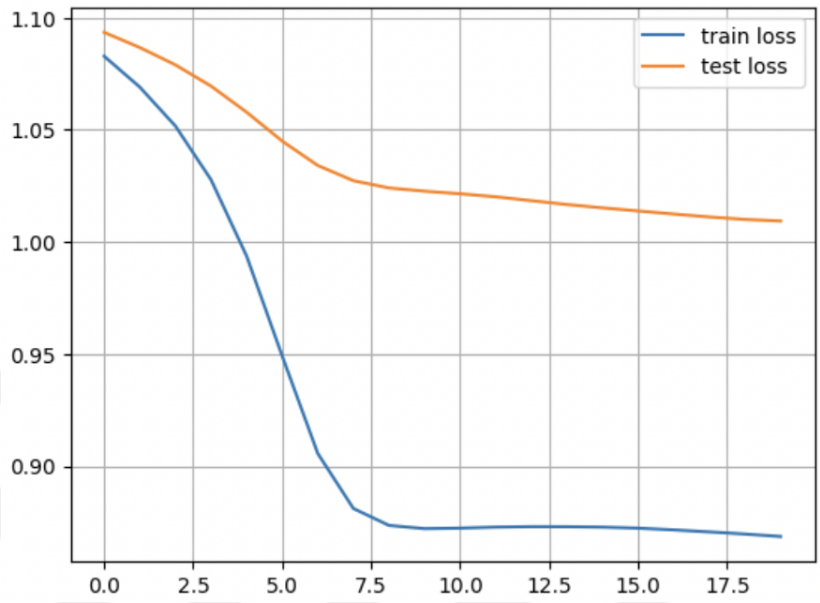

Figure 6.3. Long Short Term Memory Algorithm Loss Graph for 3 Labels with 10-Second Data.

6.2.4. 2-Label Experiment with 10-Second Sessions to Predict Minute Label

To experiment with how our LSTM algorithm predicts minute-wise labels, we again generated 10-second-windowed data from the data collected on Harbiye - Unkapanı direction but relabeled it as ”safe” and ”danger” to see how dangerous behaviors would be detected. We used the parameters listed below for the LSTM algorithm and implemented it on all sessions with 10-second counts of Harbiye - Unkapanı data. We took the sequence as 6 to be able to divide each minute as sequence.

- Input Size: 50
- Sequence Length: 6
- Number of Layers: 2
- Hidden Size: 256
- Number of Classes: 2
- Learning Rate: 0.001

- Batch Size: 1000
- Number of Epochs: 15

With this all-seconds-included set of data that cannot be divided by sessions, we got 90.71 % accuracy on the train and 90.94 % on the test. We compared predicted and actual values for each session because we were able to see all minute-wise predictions for all sessions, we knew how many minutes each session took, and the data was not randomized. We calculated the accuracy for each session and got the mean value of them. In the end, we were able to get 91.40 % accuracy. We can see the loss graph below.

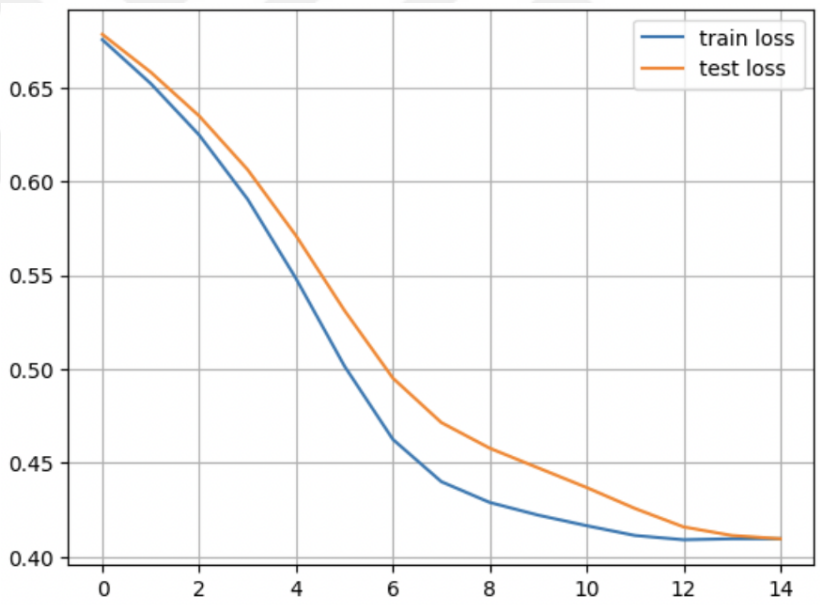

Figure 6.4. Long Short Term Memory Algorithm Loss Graph for 2 Labels with 10-Second Data.

6.2.5. 3-Label Experiment with 9-Minute Session to Predict Session Label

The LSTM algorithm is useful for time series data and we can get better results if we provide the minutes of a session as one sequence to it. We implemented our experiment finally with 9 minutes of each session labeled as normal, aggressive, and

risky and took the sequence length as 9. We combined the two directions as Harbiye - Unkapanı and Unkapanı - Harbiye as well. We used the parameters listed below.

- Input Size: 50
- Sequence Length: 9
- Number of Layers: 2
- Hidden Size: 256
- Number of Classes: 3
- Learning Rate: 0.0001
- Batch Size: 100
- Number of Epochs: 165

With this 9-minute 3-label data set, we got 71.91 % accuracy on the train and 58.97 % on the test. We can see the loss graph below.

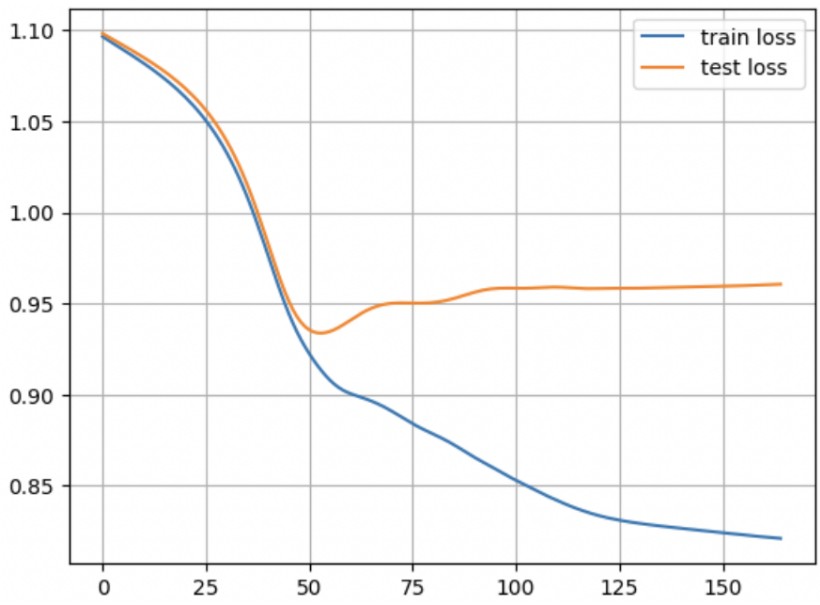

Figure 6.5. Long Short Term Memory Algorithm Loss Graph for 3 Labels with 9-Minute Data.

6.2.6. 2-Label Experiment with 9-Minute Session to Predict Session Label

With our focus to detect really dangerous behaviors, we implemented our experiment again with 9 minutes of each session combined in total but this time relabeled as "safe" and "danger". We combined the two directions as Harbiye - Unkapanı and Unkapanı - Harbiye as well. We used the parameters listed below.

- Input Size: 50
- Sequence Length: 9
- Number of Layers: 2
- Hidden Size: 256
- Number of Classes: 2
- Learning Rate: 0.0001
- Batch Size: 100
- Number of Epochs: 185

With this 9-minute 2-label data set, we got 93.26 % accuracy on the train and 92.31 % on the test.

We can see the loss graph below.

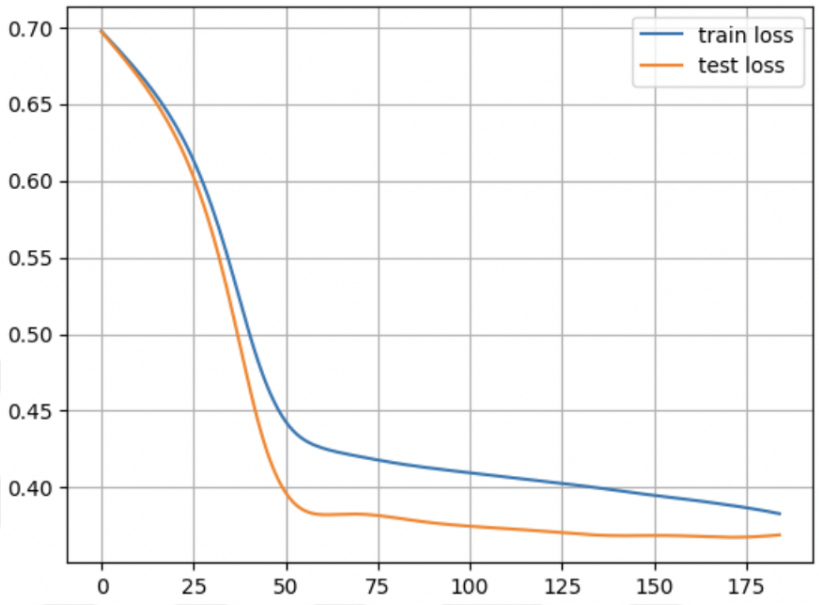

Figure 6.6. Long Short Term Memory Algorithm Loss Graph for 2 Labels with 9-Minute Data.

6.3. Federated Learning

With the process we applied as explained in Chapter 5, we combined our LSTM algorithm with federated learning architecture. We divided our total 9 minutes of Harbiye - Unkapanı and Unkapanı - Harbiye session data into train and test sets. Then, we divided the train part into 3 sets to have 3 different client models. We applied the experiment on 2-label data set. The number of sessions included in each client was 30, 30, and 29. We reached the accuracy of 96.67 %, 93.33 % and 93.10 % for them in training part respectively. After summing these vectors of parameters of these models, we updated the global model and applied it to the test data set with 39 sessions. We got 94.87 % accuracy in the end, which is a successful result to detect dangerous driving behavior.

6.4. Testing on Driver Data

In this study, we suggested applying our algorithm to the data we collected from passengers while they were traveling in a bus, and shared our results on this data. We developed a new algorithm and got successful results on our own real-time primary passenger data. Additionally, we implemented our algorithm on driver data existing in the literature. We ultimately aimed to test how our algorithm would perform on a secondary data used previously in the literature.

We selected the data collected by Cojocaru et al. [30, 31] because it was a recent data set collected the previous year and is similar to our structure. Similar to our study, they focused on Accelerometer and Gyroscope sensors but with 2 samples per second rate. The labels they used were slow, normal, and aggressive. The smartphone they used to have sensors was Samsung Galaxy S21. Because they did not have many samples collected, we combined train and test data sets they shared, which was already suggested in their study. Then, we divided the total set as 0.7 train and 0.3 test. We applied our feature generation functions shared in Section 4.3 to prepare the data and then applied our LSTM algorithm with 9-minute-windowed sequences.

We used the parameters listed below.

- Input Size: 50
- Sequence Length: 9
- Number of Layers: 2
- Hidden Size: 256
- Number of Classes: 2 for 2 label, 3 for 3 label
- Learning Rate: 0.0001
- Batch Size: 10
- Number of Epochs: 90 for 2 label, 55 for 3 label

6.4.1. 3-Label Experiment with 9-Minute Sequences for the Secondary Data Set

Firstly, we used the data set as it was, with 3 labels as slow, normal, and aggressive. The accuracy rate we got was 75.00 % on the train and 100.00 % on the test. We can see the loss graph below.

When we compared this result with the findings they shared in their study, we saw that the accuracy rates increased with our algorithm. The rates they found with 3-label classifications were 59.26 % and 63.70 % in their different papers [30, 31].

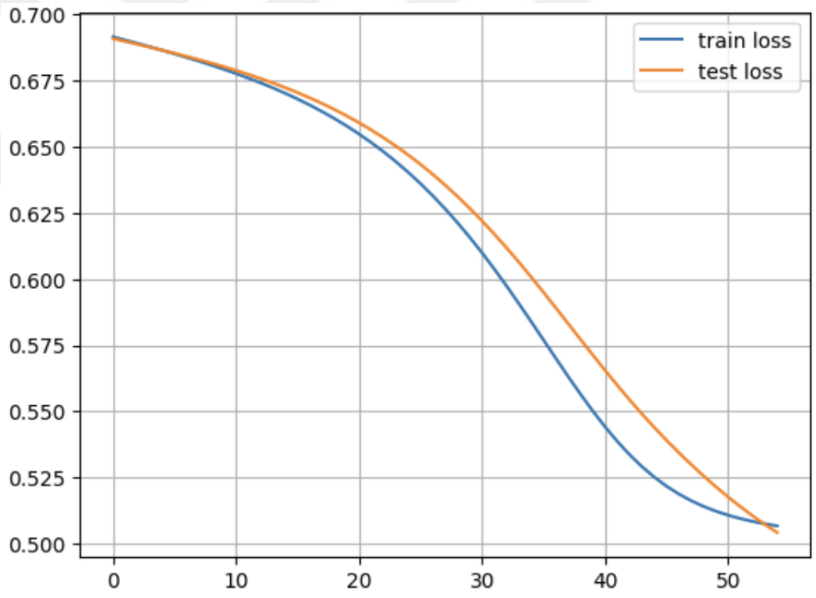

Figure 6.7. Loss Graph for 3-Label 9-Minute Sequence Driver Data.

6.4.2. 2-Label Experiment with 9-Minute Sequences for the Secondary Data Set

Secondly, we used the data set with 2 labels (danger and safe) with danger for aggressive, and safe for slow and normal. The accuracy rate we got was 100.00 % on the train and 100.00 % on the test. We can see the loss graph below.

When we compared this result with the findings they shared in their study, we observed that the accuracy rates increased with our algorithm. The rates they found with 2-label classification were 79.50 % and 91.94 % in their different papers [30, 31].

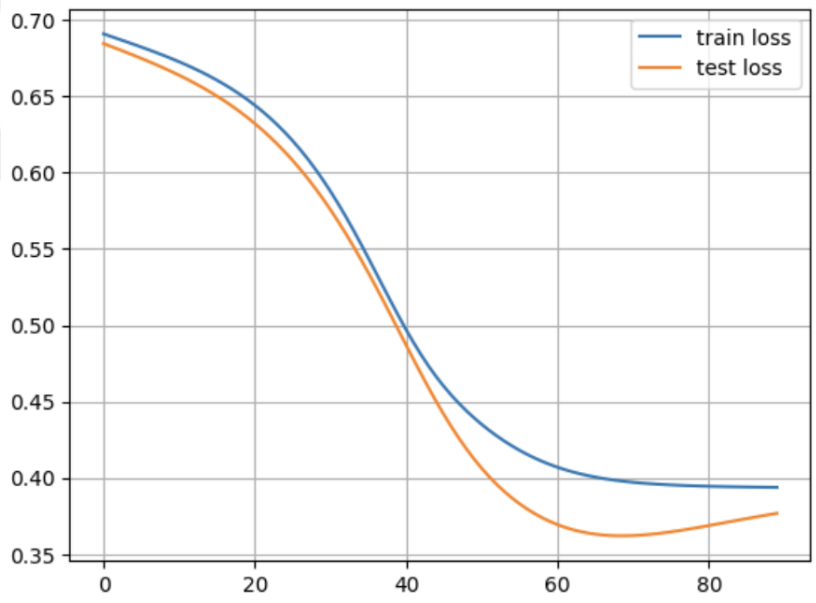

Figure 6.8. Loss Graph for 2-Label 9-Minute Sequence Driver Data.

7. RESULTS AND DISCUSSIONS

We applied 2 different machine learning algorithms and 6 different LSTM algorithms in our experiment. We wanted to see if it makes a difference in accuracy if the input is classified differently.

Our algorithm was more successful in classifying 2-label data than classifying 3-label data. 2-label classification was enough for detecting risky driving behaviors. When we compared the machine learning algorithms with our LSTM algorithm, the former were also more successful in 2-label classifications than 3-label classifications.

In general, we see that we can increase the accuracy of machine learning algorithms with our Long Short Term Memory and Federated Learning algorithms.

We were expecting a 9-minute session-wise prediction to be successful to detect our time series data. It was successful with the 2-label classification, but the 3-label classification was better for the experiment with all minutes of the data included. Because we prepared probability shapes for the labels, having successful results with undivided proper sessions was expected.

We see that it also gives successful results with 2-label prediction when we give 10 seconds of each session to predict minute-wise labels. This explains that we can get real-time feedback during the session without waiting until the end of the session. We can get feedback each minute.

Our federated learning algorithm also gives high-accuracy results for each client. The client-wise results increased up to 96.67 % during training. It also resulted in 94.87 % during testing. This means that the client-distributed model can predict the driving behavior successfully and it also increases the accuracy.

8. CONCLUSION

8.1. Remarks

In this study, we aimed to use smartphone sensor data to detect driving behavior automatically and get the results from a passenger perspective via the application we developed. With this aim, we collected the data of 128 sessions which had 64 sessions for each direction on a path that takes approximately 15 minutes to drive. This data set also included the voting results of the passengers for each driving session. Passengers used our application on their smartphones to record the accelerometer and gyroscope sensor data during driving sessions and recorded their votes in the application as well.

Firstly, we applied machine learning algorithms to our data set and got accuracy results between 82 % to 93 %. Even if these algorithms were resulting in successful accuracy, we needed a deep learning algorithm to classify session-wise time series input.

The deep learning algorithm, Long Short Term Memory in our study, which is a branch of Recurrent Neural Network algorithms gave successful results for our data set collected from passengers and can classify the driving behavior in accordance with the session-wise time series data. We got 92.31 % accuracy in the test for 9-minute sessions classified as "safe" or "danger".

Because we believed that developing a federated learning algorithm can facilitate keeping the data private and update a global model from each user's smartphone, we used our LSTM algorithm for 2-label classification with 9-minute sequenced sessions in the federated learning algorithm we developed. We updated a global model with 3 different client models and reached 94.87 % accuracy in the test.

8.2. Future Work

The results from our study can be applied to the mobile application that can give real-time feedback according to driving behavior. Thanks to our federated learning algorithm, user data can be stored locally and a global model can be trained with each session.

The results can be helpful for parents of students when they get on school buses to detect if they are experiencing safe driving behavior or not, municipalities to supervise if public transportation drivers are driving safely and general directorate of highways to find risky path directions.

REFERENCES

1. Chan, T. K., C. S. Chin, H. Chen, and X. Zhong, "A Comprehensive Review of Driver Behavior Analysis Utilizing Smartphones", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 21, No. 10, pp. 4444–4475, 2019.

2. Bengler, K., K. Dietmayer, B. Farber, M. Maurer, C. Stiller, and H. Winner, "Three Decades of Driver Assistance Systems: Review and Future Perspectives", *IEEE Intelligent Transportation Systems Magazine*, Vol. 6, No. 4, pp. 6–22, 2014.

3. Guo, H. Y., Y. Ji, T. Qu, and H. Chen, "Understanding and Modeling the Human Driver Behavior Based on MPC", *IFAC Proceedings Volumes*, Vol. 46, pp. 133–138, Seoul, Korea, 2013.

4. Singh, S., "Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey", *National Highway Traffic Safety Administration Traffic Safety Facts (Crash and Stats)*, 2018.

5. Martinez, C. M., M. Heucke, F. Wang, B. Gao, and D. Cao, "Driving Style Recognition for Intelligent Vehicle Control and Advanced Driver Assistance: A Survey", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 19, No. 3, pp. 666–676, 2017.

6. Faisal, I. A., T. W. Purboyo, and A. S. Ansori, "A Review of Accelerometer Sensor and Gyroscope Sensor in IMU Sensors on Motion Capture", *Journal of Engineering and Applied Sciences*, Vol. 15, No. 3, pp. 826–829, 2019.

7. Li, F., H. Zhang, H. Che, and X. Qiu, "Dangerous Driving Behavior Detection Using Smartphone Sensors", *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1902–1907, Auckland, New Zealand, 2016.

8. Saiprasert, C., and W. Pattara-Atikom, "Smartphone Enabled Dangerous Driving

Report System", *IEEE 46th Hawaii International Conference on System Sciences*, pp. 1231–1237, Wailea, Maui, Hawaii, USA, 2013.

9. Alluhaibi, S. K., M. S. Al-Din, and A. Moyaid, "Driver Behavior Detection Techniques: A Survey", *International Journal of Applied Engineering Research*, Vol. 13, No. 11, pp. 8856–8861, 2018.

10. Chhabra, R., S. Verma, and C. R. Krishna, "A Survey on Driver Behavior Detection Techniques for Intelligent Transportation Systems", *IEEE 7th International Conference on Cloud Computing, Data Science and Engineering Confluence*, pp. 36–41, Noida, India, 2017.

11. Al-Sultan, S., A. H. Al-Bayatti, and H. Zedan, "Context Aware Driver Behavior Detection System in Intelligent Transportation Systems", *IEEE Transactions on Vehicular Technology*, Vol. 62, No. 9, pp. 4264–4275, 2013.

12. Azadani, M. N., and A. Boukerche, "Driving Behavior Analysis Guidelines for Intelligent Transportation Systems", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 23, No. 7, pp. 6027–6045, 2021.

13. Vasconcelos, I., R. O. Vasconcelos, B. Olivieri, M. Roriz, M. Endler, and M. C. Junior, "Smartphone Based Outlier Detection: A Complex Event Processing Approach for Driving Behavior Detection", *Journal of Internet Services and Applications*, Vol. 8, No. 1, pp. 1–30, 2017.

14. Zhao, L., F. Yang, L. Bu, S. Han, G. Zhang, and Y. Luo, "Driver Behavior Detection via Adaptive Spatial Attention Mechanism", *Advanced Engineering Informatics*, Vol. 48, p. 101280, 2021.

15. Kang, H. B., "Various Approaches for Driver and Driving Behavior Monitoring: A Review", *In Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 616–623, Sydney, Australia, 2013.

16. Ghandour, R., A. J. Potams, I. Boulkaibet, B. Neji, and Z. A. Barakeh, "Driver Behavior Classification System Analysis Using Machine Learning Methods", *Applied Sciences*, Vol. 11, No. 22, p. 10562, 2021.

17. Di Giacomo, U., R. Casolare, O. Eigner, F. Martinelli, F. Mercaldo, T. Priebe, and A. Santone, "Exploiting Supervised Machine Learning for Driver Detection in a Real World Environment", *Procedia Computer Science*, Vol. 192, pp. 2440–2449, 2021.

18. Sharma, V., H. C. Chen, and R. Kumar, "Driver Behaviour Detection and Vehicle Rating Using Multi-UAV Coordinated Vehicular Networks", *Journal of Computer and System Sciences*, Vol. 86, pp. 3–32, 2017.

19. Shahverdy, M., M. Fathy, R. Berangi, and M. Sabokrou, "Driver Behaviour Detection Using 1D Convolutional Neural Networks", *Electronics Letters*, Vol. 57, No. 3, pp. 119–122, 2021.

20. Shahverdy, M., M. Fathy, R. Berangi, and M. Sabokrou, "Driver Behavior Detection and Classification Using Deep Convolutional Neural Networks", *Expert Systems with Applications*, Vol. 149, p. 113240, 2020.

21. Al-Hussein, W. A., L. Y. Por, M. L. Kiah, and B. B. Zaidan, "Driver Behavior Profiling and Recognition Using Deep Learning Methods: In Accordance with Traffic Regulations and Experts Guidelines", *International Journal of Environmental Research and Public Health*, Vol. 19, No. 3, p. 1470, 2022.

22. Alkinani, M. H., W. Z. Khan, and Q. Arshad, "Detecting Human Driver Inattentive and Aggressive Driving Behavior Using Deep Learning: Recent Advances, Requirements and Open Challenges", *IEEE Access*, Vol. 8, pp. 105008–105030, 2020.

23. Mumcuoglu, M. E., G. Alcan, M. Unel, O. Cicek, M. Mutluergil, M. Yilmaz,

and K. Koprubasi, "Driving Behavior Classification Using Long Short Term Memory Networks", *IEEE AEIT International Conference of Electrical and Electronic Technologies for Automotive*, pp. 1–6, Torino, Italy, 2019.

24. Cura, A., H. Kucuk, E. Ergen, and I. B. Oksuzoglu, "Driver Profiling Using Long Short Term Memory (LSTM) and Convolutional Neural Network (CNN) Methods", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 22, No. 10, pp. 6572–6582, 2020.

25. Kadri, N., A. Ellouze, M. Ksantini, and S. H. Turki, "New LSTM Deep Learning Algorithm for Driving Behavior Classification", *Cybernetics and Systems*, Vol. 54, No. 4, pp. 387–405, 2023.

26. Saleh, K., M. Hossny, and S. Nahavandi, "Driving Behavior Classification Based on Sensor Data Fusion Using LSTM Recurrent Neural Networks", *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–6, Yokohama, Japan, 2017.

27. Kouchak, S. M., and A. Gaffar, "Using Bidirectional Long Short Term Memory with Attention Layer to Estimate Driver Behavior", *IEEE 18th International Conference on Machine Learning and Applications (ICMLA)*, pp. 315–320, Boca Raton, Florida, USA, 2019.

28. Zhang, H., Z. Nan, T. Yang, Y. Liu, and N. Zheng, "A Driving Behavior Recognition Model with Bi-LSTM and Multi-Scale CNN", *IEEE Intelligent Vehicles Symposium (IV)*, pp. 284–289, Las Vegas, Nevada, USA, 2020.

29. Khodairy, M. A., and G. Abosamra, "Driving Behavior Classification Based on Oversampled Signals of Smartphone Embedded Sensors Using an Optimized Stacked-LSTM Neural Networks", *IEEE Access*, Vol. 9, pp. 4957–4972, 2021.

30. Cojocaru, I., and P. S. Popescu, "Building a Driving Behaviour Dataset", RoCHI,

2022.

31. Cojocaru, I. and P. S. P. and, C. Mihaescu, "Driver Behaviour Analysis Based on Deep Learning Algorithms", RoCHI, 2022.

32. Huang, T., R. Fu, and Y. Chen, "Deep Driver Behavior Detection Model Based on Human Brain Consolidated Learning for Shared Autonomy Systems", *Measurement*, Vol. 179, p. 109463, 2021.

33. Brahim, S. B., H. Ghazzai, H. Besbes, and Y. Massoud, "A Machine Learning Smartphone Based Sensing for Driver Behavior Classification", *IEEE International Symposium on Circuits and Systems*, pp. 610–614, Austin, Texas, USA, 2022.

34. Bahadoor, K., and P. Hosein, "Application for the Detection of Dangerous Driving and an Associated Gamification Framework", *IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pp. 276–281, Vienna, Austria, 2016.

35. Li, Y., F. Xue, L. Feng, and Z. Qu, "A Driving Behavior Detection System Based on a Smartphone's Built in Sensor", *International Journal of Communication Systems*, Vol. 30, No. 8, p. e3178, 2017.

36. Kaplan, S., M. A. Guvensan, A. G. Yavuz, and Y. Karalurt, "Driver Behavior Analysis for Safe Driving: A Survey", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 16, No. 6, pp. 3017–3032, 2015.

37. Chhabra, R., S. Singh, and V. Khullar, "Privacy Enabled Driver Behavior Analysis in Heterogeneous IoV Using Federated Learning", *Engineering Applications of Artificial Intelligence*, Vol. 120, p. 105881, 2023.

38. Doshi, K., and Y. Yilmaz, "Federated Learning Based Driver Activity Recognition for Edge Devices", *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3338–3346, New Orleans, Los Angeles, USA,

2022.

39. Vyas, J., D. Das, and S. Chaudhury, "Federated Learning Based Driver Recommendation for Next Generation Transportation System", *Expert Systems with Applications*, p. 119951, 2023.

40. Dillon, K. M., and D. L. Dunn, "Passenger Complaints about Driver Behaviors", *Accident Analysis and Prevention*, Vol. 37, No. 6, pp. 1012–1018, 2005.

41. Zhang, N., P. Cheng, P. Ning, D. C. Schwebel, and G. Hu, "Conflicts between Bus Drivers and Passengers in Changsha China", *Accident Analysis and Prevention*, Vol. 169, p. 106623, 2022.

42. Ramage, R., and A. Howley, "Parents' Perceptions of the Rural School Bus Ride", *The Rural Educator*, Vol. 27, No. 1, pp. 15–20, 2005.

43. Bonaccorso, G., *Machine Learning Algorithms*, Packt Publishing Ltd., Birmingham, 2017.

44. Shinde, P. P., and S. Shah, "A Review of Machine Learning and Deep Learning Applications", *Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pp. 1–6, Pune, India, 2018.

45. Mahesh, B., "Machine Learning Algorithms - A Review", *International Journal of Science and Research (IJSR)*, Vol. 9, pp. 381–386, 2020.

46. Neto, J. A., "A16 — Do You Know What Machine Learning is? (Summary)", 2021, `https://medium.com/xnewdata/a16-introduction-to-machine-learning-summary-b849393da185`, accessed on May 1, 2023.

47. Wahab, A., "Top 8 Machine Learning Algorithms Explained in Less Than 1 Minute Each", 2022, `https://datasciencedojo.com/blog/machine-learning-algorithms-explanation/`, accessed on May 1, 2023.

48. "Machine Learning Algorithms", https://www.javatpoint.com/machine-learning-algorithms, accessed on May 1, 2023.

49. Schmidhuber, J., "Deep Learning", 2015, http://www.scholarpedia.org/article/Deep_Learning, accessed on May 3, 2023.

50. Nielsen, M. A., *Neural Networks and Deep Learning*, Determination Press, San Francisco, 2015.

51. Grieve, P., "Deep Learning vs. Machine Learning: What's the Difference?", 2023, https://www.zendesk.com/blog/machine-learning-and-deep-learning, accessed on May 9, 2023.

52. Bengio, Y., I. Goodfellow, and A. Courville, *Deep Learning*, MIT Press, Cambridge, 2017.

53. Sharma, A., "Differences between Machine Learning Deep Learning", 2018, https://www.datacamp.com/tutorial/machine-deep-learning, accessed on May 9, 2023.

54. Mishra, C., and D. L. Gupta, "Deep Machine Learning and Neural Networks: An Overview", *IAES International Journal of Artificial Intelligence*, Vol. 6, No. 2, p. 66, 2017.

55. Anderson, D., and G. McNeill, *Artificial Neural Networks Technology*, Kaman Sciences Corporation, New York, 1992.

56. Medsker, L., and L. C. Jain, *Recurrent Neural Networks: Design and Applications*, CRC Press, Washington, 1999.

57. Safshine, A., and S. Amidi, "Recurrent Neural Networks Cheatsheet", https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks, accessed on May 10, 2023.

58. Pattanayak, S., *Pro Deep Learning with TensorFlow*, APress, Karnataka, 2017.

59. Greff, K., R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey", *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 28, No. 10, pp. 2222–2232, 2017.

60. Graves, A., "Generating Sequences with Recurrent Neural Networks", ArXiv:1308.0850, 2013.

61. Kairouz, P., H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. D'Oliveira, H. Eichner, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascon, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecni, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Ozgur, R. Pagh, H. Qi, D. Ramage, R. Raskar, M. Raykova, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramer, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and Open Problems in Federated Learning", *Foundations and Trends in Machine Learning*, Vol. 14, No. 1, pp. 1–210, 2021.

62. Konecni, J., H. B. McMahan, D. Ramage, and P. Richtarik, "Federated Optimization: Distributed Machine Learning for On-Device Intelligence", ArXiv:1610.02527, 2016.

63. "Motion Sensors", `https://developer.android.com/guide/topics/sensors/sensors_motion`, accessed on May 10, 2023.

64. Garcia-Ceja, E., V. Osmani, and O. Mayora, "Automatic Stress Detection in Working Environments from Smartphones' Accelerometer Data: A First Step", *IEEE Journal of Biomedical and Health Informatics*, Vol. 20, No. 4, pp. 1053–1060, 2015.

65. Brownlee, J., "How to Use StandardScaler and MinMaxScaler Transforms

in Python", 2020, https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/, accessed on May 10, 2023.

66. Xiao, Y., and Y. Yin, "Hybrid LSTM Neural Network for Short Term Traffic Flow Prediction", *Information*, Vol. 10, No. 3, p. 105, 2019.

67. Salehinejad, H., S. Sankar, J. Barfett, E. Colak, and S. Valaee, "Recent Advances in Recurrent Neural Networks", ArXiv:1801.01078, 2017.

68. Canziani, A., "Architecture of RNN and LSTM Model", https://atcold.github.io/pytorch-Deep-Learning/en/week06/06-3/, accessed on May 10, 2023.

69. Giordano, D., "7 Tips to Choose the Best Optimizer", 2020, https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e, accessed on May 11, 2023.