PhD-FSTC-2020-03
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defence held on 09/01/2020 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Sasan JAFARNEJAD
Born on 14 December 1988 in Tehran (Iran)

Machine Learning-based Methods for Driver Identification and Behavior Assessment: Applications for CAN and Floating Car Data

Dissertation defence committee

“If you wish to improve, be content to appear clueless or stupid in extraneous matters—don’t wish to seem knowledgeable. And if some regard you as important, distrust yourself.”

Epictetus, Enchiridion, 13a

Abstract

The exponential growth of car generated data, the increased connectivity, and the advances in artificial intelligence (AI), enable novel mobility applications. This dissertation focuses on two use-cases of driving data, namely distraction detection and driver identification (ID). Low and medium-income countries account for 93% of traffic deaths [1]; moreover, a major contributing factor to road crashes is distracted driving [2]. Motivated by this, the first part of this thesis explores the possibility of an easy-to-deploy solution to distracted driving detection. Most of the related work uses sophisticated sensors or cameras, which raises privacy concerns and increases the cost. Therefore a machine learning (ML) approach is proposed that only uses signals from the CAN-bus and the inertial measurement unit (IMU). It is then evaluated against a hand-annotated dataset of 13 drivers and delivers reasonable accuracy. This approach is limited in detecting short-term distractions but demonstrates that a viable solution is possible. In the second part, the focus is on the effective identification of drivers using their driving behavior. The aim is to address the shortcomings of the state-of-the-art methods. First, a driver ID mechanism based on discriminative classifiers is used to find a set of suitable signals and features. It uses five signals from the CAN-bus, with hand-engineered features, which is an improvement from current state-of-the-art that mainly focused on external sensors. The second approach is based on Gaussian mixture models (GMMs), although it uses two signals and fewer features, it shows improved accuracy. In this system, the enrollment of a new driver does not require retraining of the models, which was a limitation in the previous approach. In order to reduce the amount of training data a Triplet network is used to train a deep neural network (DNN) that learns to discriminate drivers. The training of the DNN does not require any driving data from the target set of drivers. The DNN encodes pieces of driving data to an embedding space so that in this space examples of the same driver will appear closer to each other and far from examples of other drivers. This technique reduces the amount of data needed for accurate prediction to under a minute of driving data. These three solutions are validated against a real-world dataset of 57 drivers. Lastly, the possibility of a driver ID system is explored that only uses floating car data (FCD), in particular, GPS data from smartphones. A DNN architecture is then designed that encodes the routes, origin, and destination coordinates as well as various other features computed based on contextual information. The proposed model is then evaluated against a dataset of 678 drivers and shows high accuracy. In a nutshell, this work demonstrates that proper driver ID is achievable. The constraints imposed by the use-case and data availability negatively affect the performance; in such cases, the efficient use of the available data is crucial.

Acknowledgements

This work was conducted at the SECAN-Lab of the University of Luxembourg's Interdisciplinary Centre for Security Reliability and Trust (SnT).

First and foremost, I would like to thank Prof. Dr. Thomas Engel for allowing me to conduct my research at SECAN-Lab and letting me explore and investigate my topics of interest freely. Second, I would like to thank Dr.-Ing. German Castignani, for his invaluable guidance, support, encouragement, and the much-needed criticism of my research. Third, I would like to thank Dr. Fabian Lanze for his constructive feedback on my work.

I want to thank my friends and the former members of the VehicularLab, especially Walter, Thierry, Lara, and my former office mates, Salvatore, Chris, and Mathis. I also thank all my colleagues at SECAN-Lab and SnT, who have been part of this journey and made the work enjoyable.

I express my deepest gratitude to my long-time friends, Hossein and Sam, for their unconditional support throughout the past four years.

Most of all, I would like to thank my family, especially my mom Robabeh, my sister Raheleh, my brothers Mohammad and Abbas, and Samira, for always supporting me and encouraging me with their best wishes.

The experiments presented in this dissertation were carried out using the HPC facilities of the University of Luxembourg [3]. Moreover, this work would not be possible without the datasets kindly provided by Motion-S S.A. and VPALAB of Sabanci University.

Contents

List of Figures

List of Tables

If you don't work on important problems, it's not likely that you'll do important work.

Richard Hamming

1 Introduction

The number of road deaths has reached 1.35 million per year in 2016 and continues to climb. It is the leading cause of death among children and young adults aged 5–29 [1]. Last year in Europe, over 25,000 lost their lives in road collisions; in the United States, that number was over 40,000. European Union (EU) is committed to Vision Zero, a multi-national road safety project aimed to achieve a road traffic system with no fatalities [4]. European Union (EU) aims to reduce road death to almost zero by 2050. Although the EU's preventive actions resulted in a 21% decrease between 2010 and 2018, this decrease has stagnated, and European countries missed their interim targets for the 2011–20 period [5]. For instance, in the United Kingdom, distraction or impairment was a contributory factor to 24% of fatal accidents in 2014, which has grown to 27% in 2017 [2]. A recent study shows that drivers use their phones in about 88% of their journeys [6]. Mobile phone increases the likelihood of a crash by fourfold [7]. Another critical contributory factor to fatal crashes is drowsy/fatigued driving. Fatigued driving increases the risk of involvement in crash or near-crash by nearly four times [8, 9]. Sleep deprivation can increase the risk of a crash by eight times [10], this should not come as a surprise since even moderate sleep deprivation impairs cognitive and motor performance, equivalent to legally prescribed levels of alcohol intoxication [11]. *Is it possible to detect distracted driving using a system that is affordable and easy to deploy?*

Drowsiness and fatigue are especially common among commercial long-haul drivers. Sleepiness/fatigue increases the involvement likelihood of a commercial vehicle in a fatal collision [12]. Many countries have put forward regulations to prevent long-haul truck drivers from overwork. In Europe, the regulation (EC) No 561/2006 [13] defines *driving time and rest periods*, and it is enforced using analog and smart tachographs introduced by directive 165/2014 [14]. Meanwhile, a poll conducted by a union uncovers that in the month before the poll, 40% of drivers had been asked to exceed their duty hours, this is even worse among self-employed drivers [15]. Another study finds that over 70% of drivers work more than 11 hours daily, and 43% indicated that "sometimes to always" violate the working hours [16]. These drivers not only risk their own lives; they also endanger other road users. There are also indications that drivers tamper with these tachographs [17]. *Is there a tamper-resistant way of enforcing these regulations? Perhaps tachographs that can recognize the drivers based on their driving style.*

Ride-sharing applications such as Uber, Lyft, Bolt, have revolutionized transportation by providing a level of comfort that once deemed unimaginable, but they have issues of their own. There have been many reports of sexual assault and various other crimes by ride-sharing drivers [18]. Drivers commit fraud or cheat the system by creating fake global positioning system (GPS) traces, initiating ride requests from stolen accounts, and more [19]. Introduction of background checks by the ride-sharing companies—now mandatory in some jurisdictions—has reduced the crime rates, but new workarounds emerge. Some drivers use stolen accounts or share the same account, which enables a myriad of problems. Similar challenges exist for on-demand delivery services such as Deliveroo, and even traditional fleets. *What if fleets were able to verify the driver's identity based on their driving style?*

These are just examples of open issues that we believe we can undertake. Several enablers make this the right time to approach these issues. Cars are no longer mechanical products with electronics in them; they are—as it is fashionable to call them—"computers on wheels." Modern cars contain in the order of hundreds of embedded processors, and this number is proliferating. This increase in the number of electronic control units (ECUs) is partly due to Moore's law [20], but more importantly, driven by the demand for the comfort provided by cyber-physical systems (CPSs). For instance, the adaptive cruise control maintains speed and longitudinal distance from the leading car, the lane-keep assist (LKA) keeps the car between the road markings. The automatic braking brings the car to a halt to avoid collisions. These cyber-physical systems (CPSs), require sensors to perceive the world and powerful processors to make sense of it and act accordingly. As a result, modern cars generate vast amounts of data (sensor measurements) and thus require massive computing power.

More recently, this vast amount of data has become accessible to third-parties. Car generated data is sent to the cloud through several means, such as the car's telematics units (2G, 3G, 4G/LTE, and 5G), on-board unit (OBU) (dedicated short-range communications (DSRC)). Additionally, third-party aftermarket data collection devices exist that add additional connectivity to cars with no telematics units. Such devices are installed in the vehicle often plugged into the OBD-II port—a mandatory diagnostics port available in most cars manufactured since 1995. These devices (e.g., insurance dongles, black boxes) upload data to their corresponding servers via a cellular connection or through a smartphone. Some manufacturers such as BMW, Mini and, Daimler provide application programming interfaces (APIs) to third-parties to access car data in order to provide services. Other companies are working to consolidate such APIs into a single platform[1].

This data abundance prepares a fertile ground for artificial intelligence (AI) applications. In particular, machine learning (ML) based methods, and more recently, deep learning (DL) thrive at problems where vast amounts of data are available. AI is the driving force behind state-of-the-art research in natural language processing (NLP) and machine translation [21–24], image classification [25–27], object detection [28], image segmentation, text-to-speech, speech recognition, recommendation systems [29], health care, and autonomous driving.

The exponential increase in the amount of data generated by cars, and ease of access to them because of connectivity, and advances in AI, grants us this unique opportunity to explore mobility applications that are not yet fully explored. Many telematics applications have been made possible from the analysis of datasets collected from cars and the usage of machine learning (ML) techniques, such as driving behavior analysis, predictive maintenance of vehicles.

In particular, we want to explore two application areas, *distracted driving detection* and *driver identification.*

1.1 Research Question

"How could we use machine learning to profile drivers reliably?"

The objective of this dissertation is to explore some of the applications of applying machine learning methods to driving data. After careful consideration of a wide range of alternatives, we decided to focus on two applications that have a high potential impact on society, particularly *distracted driving detection* and *driver identification.*

First, we explore the possibility to detect distracted driving only using driving behavior signals. *Is it possible to detect distracted driving using driving behavior signals?* Knowing that every year, 1.35 million lose their lives in traffic crashes [1], and the fact that up to 27% of crashes are partially caused by distractions [2], we establish that there is a need for distracted driving detection solutions. However, the state-of-the-art mostly focuses on solutions based on camera [30–33] or physiological responses [34]. The problem with the camera-based systems is twofold: *a*) legal concerns due to the privacy issues and

[1]such as `https://High-Mobility.com`

compliance with the general data protection regulation (GDPR) *b)* user acceptance, the presence of cameras induce a degree of discomfort. Moreover, privacy-conscious individuals resist such systems. The physiological systems often require sensors to be attached to the body, which renders them impractical and intrusive. Moreover, these approaches require dedicated sensors, which incurs additional cost and limits their deployment. Since high-income countries only constitute 7% of road traffic deaths [1, pg. 7] it is essential to propose an affordable solution. Therefore to avoid expensive sensors and maximize ease of deployment, the ideal solution would be to use commonly available driving signals from the car's internal network (e.g., controller area network (CAN)-bus).

Second, we dedicate a more significant portion of this thesis to find out *how may we most effectively identify drivers based on their driving behavior?* The prevalence of connected vehicles and new mobility paradigms have propelled the need for accurately identifying who is behind the steering wheel on any driving situation. Depending on the use-case and available data, driver identification (ID) can be either used to replace traditional authentication methods—such as fingerprint readers—or as a secondary identification method in conjunction with the conventional approaches. Driver ID is an essential building block for new smart mobility services such as dynamic pricing for insurance, customization of comfort features, and pay-as-you-drive services. However, state-of-the-art solutions are inadequate. Moreover, since there is no high-quality public dataset suitable for driver ID, current literature is somewhat fragmented. Each work often uses signals with sample-rates that are not accessible by the rest of the research community; therefore, the results are not comparable. Common issues in the literature are the unrealistic set of signals that are not available in standard cars on the market [35–39] or use too many signals, which limits deployment [40, 41] or some works mistakenly model the state of the car rather than the driver [42–45]. Complex preprocessing that often requires hand annotation of data [46, 47], scalability issues [37–40, 42, 44, 48], and the need for large amounts of data for training and prediction [49, 50]. In the second part of the thesis, we aim to narrow the gap and address the issues mentioned above.

1.2 Methodology

Data is essential to any ML approach. The first concern is to obtain the relevant driving datasets. There are two general categories of driving data: *a) CAN-data*, often high sample-rate, usually collected from car's internal network (e.g., CAN-bus), OBD-II dongles. *b) FCD* provided by global navigation satellite systems (GNSSs), usually low sample-rate, obtained from smartphones, navigation systems, or external receivers. Driver ID or distracted driving detection problems are not suitable for simulation studies. We may be able to simulate the driving environment, but we cannot yet adequately simulate the human driver. Therefore we conduct our studies using real-life datasets.

The first dataset that we used for four out of five studies in this work is a high sample-rate real-life dataset of 105 drivers called Uyanik, which contains data from CAN-bus, inertial measurement unit (IMU), and video feeds [51].

In this work, we aim to detect *distracted driving* and *identify drivers* using driver behavior signals. These two problems are similar, and we can model them as a sequence classification task. We use a sliding window approach due to its simplicity and flexibility. In this approach, the sequential data is broken into smaller overlapping windows called a *frame*, and then each piece is considered as a separate example.

To study distracted driving detection, we require an annotated dataset. In the Uyanik dataset, participants perform secondary cognitive tasks while driving; we use this to create our dataset. We use driver-facing videos to manually annotate a subset of the dataset consisting of 13 drivers. We propose an ML approach based on discriminative classifiers to detect driver distraction using data from CAN-bus and IMU.

We explore driver ID in more depth and use an incremental approach to address the shortcomings of state-of-the-art methods. We propose three approaches using CAN-data, all based on the sliding window approach. In the first study, we focus on finding the minimal set of signals and features that provides adequate identification performance, at the same time, are available on most cars. In the second study, in order to improve scalability and improve identification performance, we propose a model based on Gaussian mixture models (GMMs), decrease the number of signals, and use a smaller feature-set. In the third study, we propose a deep learning approach to driver ID, which we name the *deep driver* model. The *deep driver* model is an architecture based on *Triplet networks* [52]. The *deep driver* encodes a *block* of driving data into a d-dimensional *embedding* vector, which later is used for driver ID. We evaluate the CAN-data based driver ID methods using the Uyanik dataset, investigating various aspects of the driver ID task such as the influence of the amount of training and prediction data on accuracy.

Lastly, we touch on driver ID using floating car data (FCD). We propose a deep learning architecture composed of embedding and recurrent neural network layers. The input is a set of features, obtained from contextualized location data-points of a trip. For the evaluation, we use a large driving dataset covering 678 drivers in the Greater Region of Luxembourg.

1.3 Contributions

The first contribution is an ML-based distraction detection mechanism. We propose a non-intrusive and privacy-friendly approach; it only uses five signals from CAN-bus and three signals from IMU. The approach is then evaluated on a dataset of 13 drivers and provides decent performance. This approach is limited in detecting short-term distractions but demonstrates the possibility of distraction detection using driving data. This contribution is presented in Chapter 4 and published in [53].

The second contribution is a driver ID approach based on discriminative classifiers. This approach only relies on five standard signals available from CAN-bus. We discover that the best features for driver ID are the spectral and cepstral features of driver-operated controls such as steering wheel and the pedals. However this approach has some limitations: *a*) the need to retrain the model after enrollment of a new driver, *b*) large amounts of training and prediction data to achieve good accuracy (20 minutes of training and 4 minutes of prediction data to obtain above 90% accuracy for 5, and 15 drivers, respectively). This contribution is presented in Chapter 6 and published in [54].

The third contribution is a driver ID algorithm based on GMM. This approach uses spectral and cepstral features from 2 CAN-bus signals, namely, the gas pedal position and the steering wheel angle. Moreover, vehicle speed (VS) can be used to estimate driving route [55], since VS is not used, this approach is more privacy-friendly than our previous contribution. The GMM approach has low complexity, produces compact models, and is scalable, can also achieve above 90% accuracy with just 5 minutes of training data and 3 minutes for prediction. This contribution is presented in Chapter 7 and published in [56].

The fourth contribution is a deep learning approach to driver ID. In this contribution, we further reduce the amount of data needed for driver ID and, at the same time, increase the accuracy of the system. The *deep driver* model is an architecture based on the *Triplet network* [52], which encodes *blocks* of driving data into d-dimensional embedding vectors. We can use these embeddings for driver ID, driver verification, and other applications such as driver clustering that can be used to infer how many drivers have driven a car. This contribution is presented in Chapter 8.

Our last contribution is a deep neural network (DNN) architecture that relies only on GPS data and succeeds in accurately identifying the driver. Although it uses an external web service to extract contextual metrics from the locations, this method stands as an end-to-end driver ID solution. We also present an efficient way of encoding location coordinates and road-network links using embeddings. This contribution is presented in Chapter 9 and currently undergoing review as a journal article.

1.4 Manuscript Structure

Chapter 2 contains the details of the Uyanik dataset, which we use to validate the approaches presented in this work (except for Chapter 9). We organize the rest of the thesis in two parts, the first part is dedicated to the topic of distracted driving detection, and the second part encompasses the work on driver ID.

Part I begins with Chapter 3, which gives an introduction to distracted driving detection and the state-of-the-art of the subject. Then in Chapter 4, we present a non-intrusive approach to detect distracted driving.

Part II starts with Chapter 5, in which we introduce the problem of driver identification, its applications, and the main building blocks of a driver identification system. Then we present a summary of state-of-the-art solutions and highlight the gaps in the current literature on the subject. Chapter 6 proposes a driver ID method using discriminative classifiers. In Chapter 7, we propose an improved scalable approach that is based on GMM. Chapter 8 aims to reduce the amount of training data required to obtain high accuracy models. It introduces a deep learning (DL) model based on the Triplet network to significantly reduce the amount of training data required for driver identification. In Chapter 9, we explore the possibility of driver identification only using low sample-rate GPS data from smartphones.

In Chapter 10, we summarize our contributions, provide a final discussion, and present future research directions.

Errors using inadequate data are much less than those using no data at all.

Charles Babbage

2 Uyanik Dataset

IN this chapter, we present the Uyanik dataset, which we use to evaluate the CAN-data approaches in this dissertation. It contains high sample-rate (up to 32Hz) sensor measurements from sources such as CAN-bus, IMU, GPS receiver. This dataset contains driving data from 105 participants driving the same route. The Uyanik dataset is collected under the shared framework of Drive-Safe Consortium (Turkey) and NEDO (Japan) international collaborative research [51][57]. The main application focus of Uyanik is driving behavior signal processing, more specifically, applications ranging from driver identification to driving environment personalization and driver assistance.

Partner universities developed and deployed sensor-equipped vehicles sharing requirements to collect data on driving behavior under various driving conditions. The Sabanci University of Turkey, under the shared framework laid jointly by the partners; Equipped a Renault Megane with various sensors to measure the dynamic state of the vehicle and its surrounding environment. Two cameras were installed to record the driver in daylight and at night using night vision. Another camera installed in front of the vehicle pointed at the road. A differential GPS receiver logs the vehicle location. Four microphones capture the conversations carried on inside the vehicle. The IMU and CAN-bus data were used to capture the vehicle dynamics and internal state of the car. Additionally, pressure sensors were installed underneath the brake and gas pedals to record pedal actuation accurately. There is also a laser rangefinder installed in front of the vehicle that covers a 180° field of view. The complete list of available sensor signals is presented in Table 2.1.

Table 2.1: Sensor Data Available in Uyanik

Channel	Source	Details
Video facing the driver	Retrofitted	15 *fps*, 480×640
Video facing the road	Retrofitted	15 *fps*, 480×640
Driver close-talking microphone	Retrofitted	16 kHz, 16-bit
Rear-view microphone	Retrofitted	16 kHz, 16-bit
Cellular phone microphone	Retrofitted	16 kHz, 16-bit
Steering wheel angle (SWA)	CAN-Bus	32 Hz, °
Steering wheel relative speed (SWRS)	CAN-Bus	32 Hz $^{\circ}s^{-1}$
Vehicle speed (VS)	CAN-Bus	32 Hz, $\mathrm{km\,h^{-1}}$
Individual wheel speeds	CAN-Bus	32 Hz $\mathrm{km\,h^{-1}}$
WSFR Wheel speed front right	CAN-Bus	32 Hz, $\mathrm{km\,h^{-1}}$
WSFL Wheel speed front left	CAN-Bus	32 Hz, $\mathrm{km\,h^{-1}}$
WSRR Wheel speed rear right	CAN-Bus	32 Hz, $\mathrm{km\,h^{-1}}$
WSRL Wheel speed rear left	CAN-Bus	32 Hz, $\mathrm{km\,h^{-1}}$
Percent gas pedal (PGP)	CAN-Bus	32 Hz, %
Engine RPM (ERPM)	CAN-Bus	32 Hz, *rpm*
Yaw rate (YR)	CAN-Bus	32 Hz, $^{\circ}s^{-1}$
Clutch state	CAN-Bus	32 Hz, 0/1 state
Reverse gear	CAN-Bus	32 Hz, 0/1 state
Brake state	CAN-Bus	32 Hz, 0/1 state
Clutch	CAN-Bus	32 Hz, 0/1 state
Brake & Gas Pedal Pressure	Retrofitted	$\mathrm{kg\,m^{-2}}$
XYZ directional accelerations	IMU	10 Hz, g
Angular rates (Roll, Pitch, Yaw)	IMU	10 Hz, $^{\circ}\mathrm{s}^{-1}$
Laser rage-finder	Retrofitted	1-2 Hz, 181°, cm

The experiments are designed to study driver behavior while performing various secondary tasks. The data collection is done in Istanbul, Turkey. It consists of a 25 km long stretch, which includes a short ride inside the university campus, a city traffic driving, motorway traffic, a dense city driving, and, finally, the way back to the point of departure. A typical trip lasts about 45 minutes. The whole journey is divided into four segments, denoting different secondary tasks: *a*) Reference Driving *b*) Query Dialogue *c*) Signboard Reading and Navigation Dialog *d*) Pure Navigational Dialog.

Details of these tasks are out of the scope of this work (refer to [51] for more details). However, it is sufficient to know the first segment is normal driving while during other segments driver is asked to perform some tasks.

The Uyanik dataset is perfectly suitable for our studies that use CAN-data. The Uyanik is designed to be used for the study of driver distraction. The participants are required to perform secondary cognitive tasks, which are real-life examples of distractions such as mobile phone use, or conversation with passengers. Moreover, the chosen route is composed of a diverse selection of road traffic situations. This is beneficial to both distraction detection and driver ID problems. Because it resembles the traffic situations in a daily commute, additionally, the large number of signals available can be used for diagnostics and to study the driving context.

2.1 Parsing Data

The Uyanik dataset, in its original conditions, is not usable. It consists of a single directory per participant (driver), each of which consists of three directories, each of these directories contains files related to audio, video and sensor measurements. The sensor measurement directory contains separate files for CAN, IMU and GPS data, and laser rangefinder. Data from CAN, IMU, and GPS are stored in text format, and there are numerous inconsistencies in formatting and corruption are frequent. Moreover, there are missing data, either from a particular sensor (e.g., GPS), is entirely or partially, an issue that is probably occurred during data recording. We take several steps to salvage as much quality data as possible. First, we use bash scripts to remove corrupted data (garbled data). Then another script is applied to some of the files with slightly different formatting to unify formatting. Lastly, a parser written in Python is used to parse each file and store it in a more convenient format. The next step is to synchronize data from each source. Files from each data source contain some timestamp, but each in different formats and arrangements, which makes it especially tricky but not impossible to put together and synchronize.

2.2 Sample-rate and Synchronization

Having the data in a more convenient format, and adequately timestamped is undoubtedly not enough. We considered to synchronize and upsample all the data sources to the most frequent source, which is the CAN-bus with 32 Hz. However, we noticed that for ten drivers, even CAN-bus is sampled at 10 Hz. Moreover, resampling and interpolating other data sources are prone to introducing noise. Regardless of the challenges and risks involved, we used the Pandas library [58] to upsample all sensor data to 32 Hz with linear interpolation. It is worth mentioning later this decision was reversed, and instead, we decided only to use drivers with 32 Hz CAN-bus data. This decision is motivated by the fact that for the majority of the works in this dissertation, the only data source is CAN-bus; therefore, there is no need to synchronize this data source with the others.

2.3 Data Quality

The Uyanik dataset is an invaluable resource for research, but it has some shortcomings. We already mentioned the issue with the sampling rates. There are more irregularities in the dataset. For instance, not all the recordings include brake and gas pedal pressure sensors, and the ones that do are frequently noisy and unusable. We use several techniques used to find quality issues in the dataset. Such as:

- Visual plots of location points on the map (Figure 2.2 shows a good example).

- Programmatically find consecutive duplicate rows and manually inspect them to ensure it is not a data recorder malfunction.
- Programmatically search for irregular timestamp values (big jumps, indicating missing data).
- Plot the sensor measurements over time and visually inspect the plot. For instance, Figure 2.5 depicts a reasonable example, in a faulty recording, all signals flatline at the same time.
- Plot histograms of signals and more closely look into irregular patterns. Examples with no variation are examples of erroneous data.

2.4 Exploratory Data Analysis

In the end, there are 57 drivers with flawless CAN-bus data at 32 Hz. Figure 2.1 shows the length of each trip. Some trips are much longer than the others; this is mainly due to the trip conditions. After checking the corresponding videos, we found that at the time of the experiment, there was heavy rain, which apparently had resulted in heavy congestion and significantly prolonged the affected trips. The median trip duration is about 47 minutes, the shortest 33 minutes, and the longest 1 hour and a half; this is evidence of how variable a daily commute experience can get, which poses further challenges.

Figure 2.2 shows an example trip from the Uyanik dataset, which is one of the best cases available in the Uyanik dataset; The GPS data is absent from recordings of many drivers. From 57 drivers with useful CAN-data, 54 of them have usable GPS data, which is quite noisy as well.

Figure 2.3 shows the histogram of some of the primary sensors from the CAN-bus. The percentage gas pedal (PGP) sensor readings range from 0 to 100, but the majority of readings are below 80. It does not follow a familiar distribution, and there is a peak around 10%, which is probably the sensor reading when the foot is resting on the pedal or a gentle gas is applied to maintain the speed. The VS is measured in $\mathrm{km\,h^{-1}}$ and ranges from $0\,\mathrm{km\,h^{-1}}$ to $174\,\mathrm{km\,h^{-1}}$, speeds over 130 only happen once, probably one of the participants got overexcited. There is a notable peak at 0, representing the vehicle at a halt, such as stopping at red-lights or waiting for the traffic to make turns. There are at least two other peaks at around $30\,\mathrm{km\,h^{-1}}$ and $76\,\mathrm{km\,h^{-1}}$, the former is probably the average speed in streets and the latter the average speed in the highway. The engine revolutions per minute (ERPM) is measured in revolutions per minute (*rpm*). This signal is quite smooth, mainly because even though it is a function of driver depressing gas pedal, but since it is combined with gear shifting and filtered by the limitations of

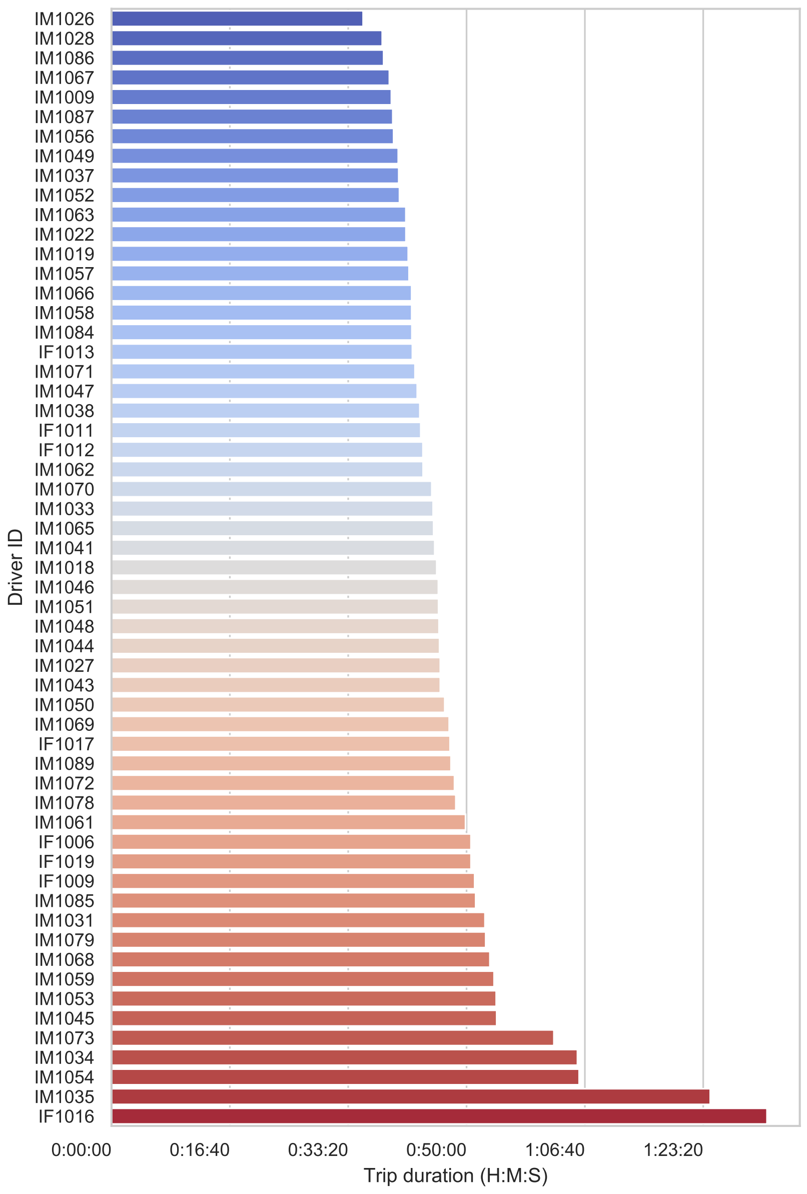


FIGURE 2.1: Trip lengths, the trips range from 33 minutes to 1 hour and a half. The median trip duration is about 47 minutes. Trips with exceptionally long duration are due to traffic jams.

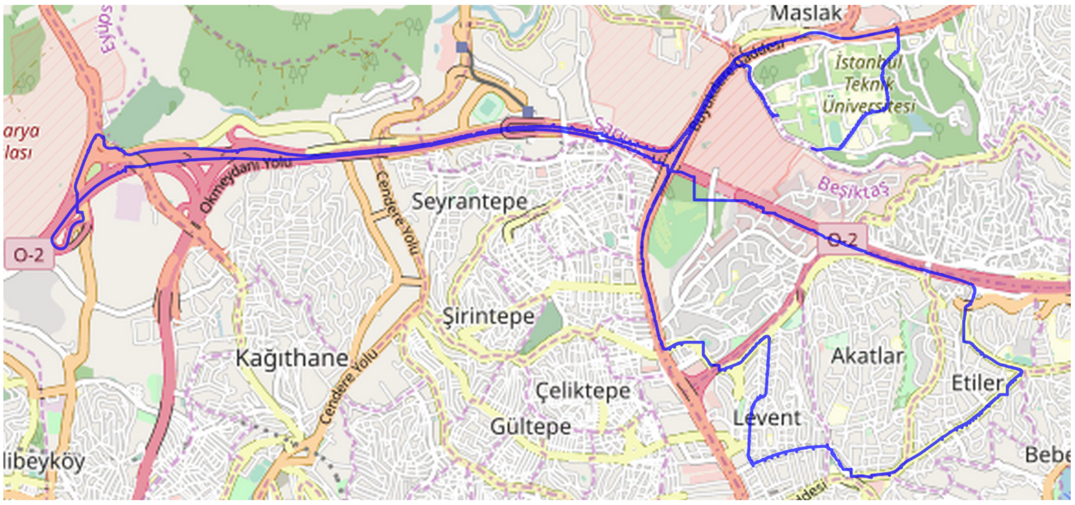


FIGURE 2.2: A trip from the Uyanik dataset. It consists of a 25 km long stretch. The trip starts at the university (top right on the map), after a short ride in the university campus, a city traffic driving, motorway traffic, a dense city driving, ends back at the point of departure.

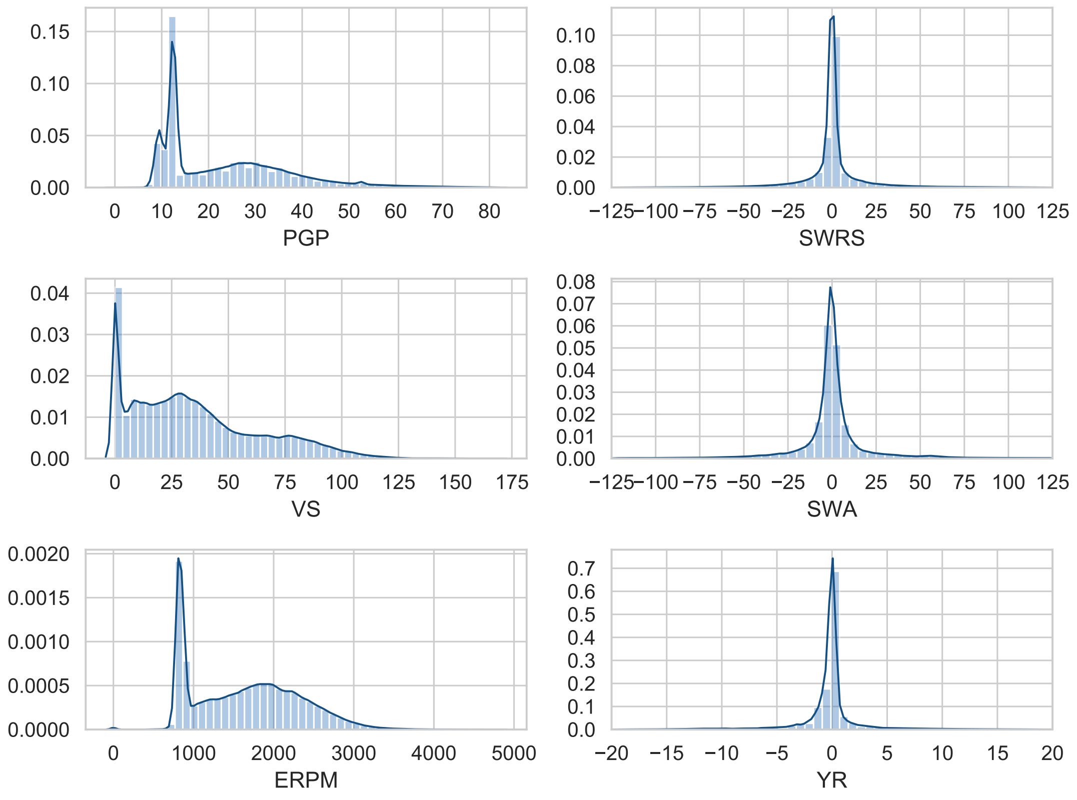


FIGURE 2.3: Histogram of main signals for all drivers - the dark blue line is drawn using kernel density estimation with Gaussian kernel, it only serves as visualization purposes. Plots for SWRS, SWA, YR are trimmed as there are a few outliers much further than the currently depicted values.

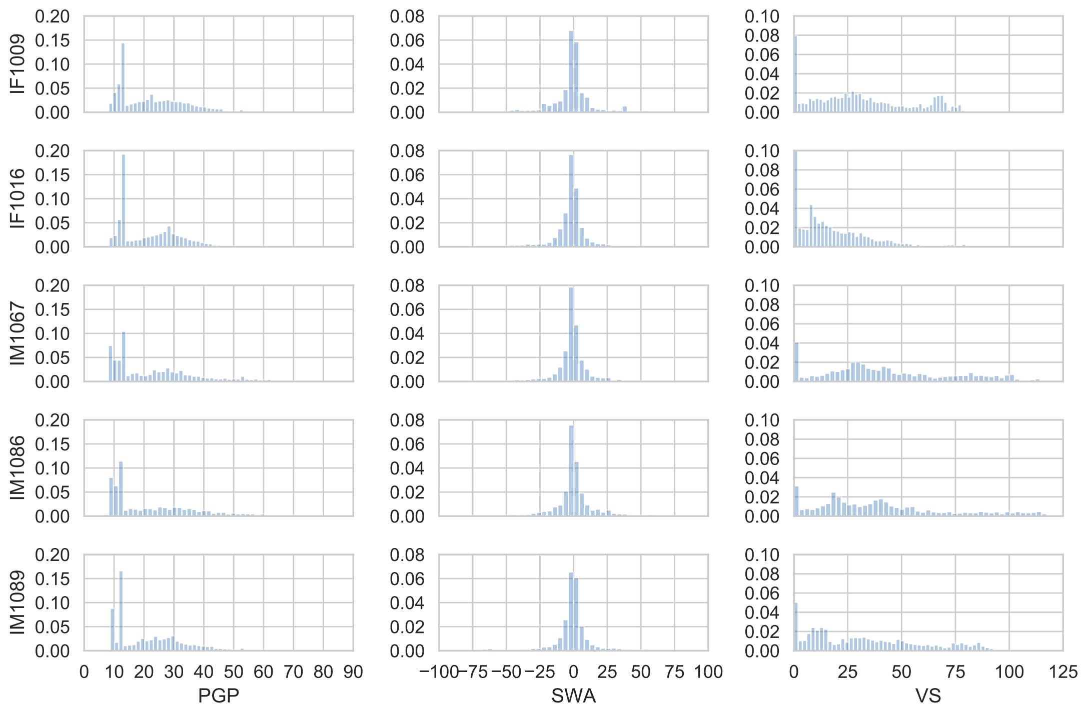


Figure 2.4: Histogram of PGP, SWA, VS for 5 drivers. To help comparison, x and y axis are in the same range for all drivers. SWA is trimmed at -100 and +100 to improve visualization.

the engine, the result is more diluted and reflects less of the driver itself. There is a peak at around 900 *rpm*, which we suspect to be the idle engine rate. The other signals presented in Figure 2.3 are steering wheel relative speed (SWRS), steering wheel angle (SWA), and yaw rate (YR). These signals have a Gaussian shape; however, these many outliers are not expected in a Gaussian distribution. The SWA is in °, and SWRS and YR are measured in $°s^{-1}$. The YR is slanted to the right and SWA to the left. This pattern is because the route is followed in a counter-clockwise direction. Therefore the majority of turns are left-turns, lead to having more negative SWA values than positive. We see the opposite pattern in YR because left turns cause positive YR.

Figure 2.4 shows how different drivers express different histograms. The most significant differences are visible in PGP and VS. For example, we see that the first two drivers always keep some pressure on the pedal. However, the other three drivers have two modes at around 10%. One probably corresponds to the idle pedal state, which is close to 10% and another peak at around 13% (a gentle continuous pressure on the pedal). It is also possible to see that the spread varies between drivers. The SWA histograms look similar to each other. We notice that the first driver has a more spread out histogram than the fourth driver. There is a clear distinction between the VS histograms. For instance, the third and fourth drivers, drive faster, but with lots of variation at higher speeds. On the other hand, the first driver seems to be more cautious and drives at the

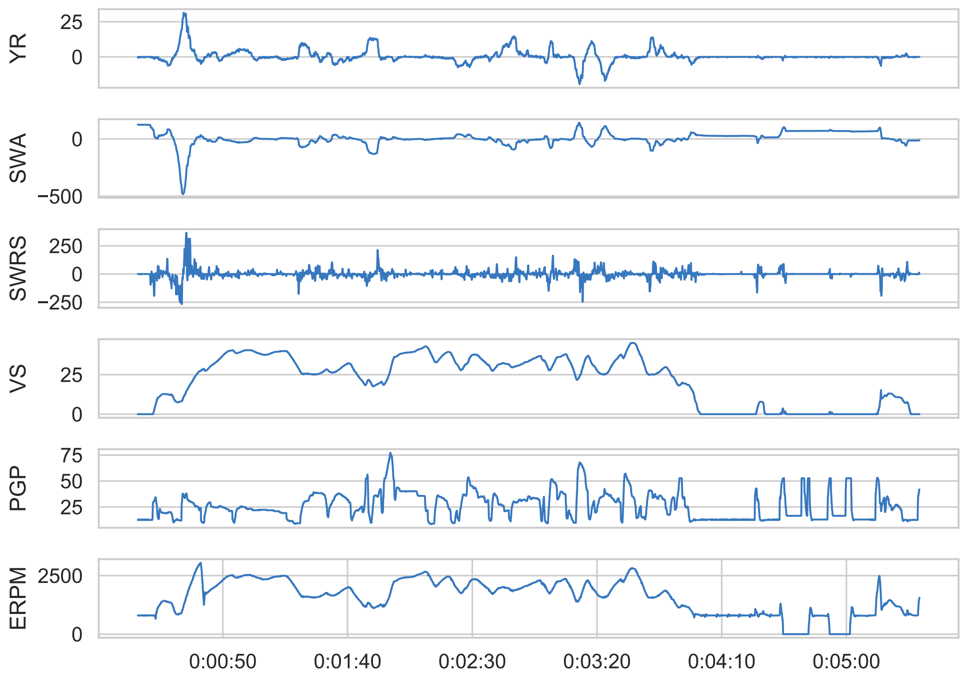


FIGURE 2.5: Example snapshot of raw sensor data for driver IM1048.

constant speed of around 70 km h^{-1} in the highway. Another interesting example would be the second driver, who is driving at lower speeds than the other drivers. One should be careful not to make judgments, only based on this histogram, because the very same driver could be a driver that usually drives much faster. However, in this recording, due to traffic conditions, they could not drive as fast as they usually would.

Figure 2.5 shows an example of raw CAN-bus data (IM1048), comprised of about 5 minutes from the beginning of the experiment. There are a few interesting things to notice in this figure. For instance, SWA and YR mirror each other. The SWRS also follows the SWA but appears to be noisy. We also notice a correlation between PGP and ERPM, which is expected. There is also a slight irregularity at about 4:30, in which ERPM drops to zero, it is likely that there is an issue, and the engine shuts off. However, since it can be seen that SWA is still following reasonable measurements, this is not a case of recording malfunction.

Part I

Distracted Driving Detection

What can be asserted without evidence can be dismissed without evidence.

Christopher Hitchens

3 Introduction to Distracted Driving Detection

NOWADAYS, Internet-enabled smartphones have become ubiquitous, and we all witness the flood of information that often arrives with a notification. We immediately divert our attention to our smartphones, even when we are behind the wheel. Statistics show that distraction-related crashes are on the rise [2]. A recent study shows that drivers use their phones in about 88% of their trips, an average of 3.5 minutes for each hour of driving [6].

Mobile phone use increases the likelihood of a crash by fourfold [7]. Numerous other research points out the dangers of distraction for passenger safety [9]. Cell phone conversations significantly reduce reaction times and have other adverse effects [59, 60]. Moreover, contrary to popular belief, the use of hands-free devices is as detrimental as hand-held devices [59–61]. Although most distractions are due to smartphone use or conversations with passengers, manual or visual distractions are as essential and require attention [62].

For years distracted driving and its repercussions have been a known threat to passenger safety, and decision-makers have been trying to address this issue, these efforts rendered futile. Currently, 150 countries have national mobile phone laws in place, and 145 of them prohibit the use of hand-held mobile phones while driving. However, there is insufficient evidence on the effectiveness of legislation [1, pg. 45].

The most effective way of reducing distraction-related crashes is replacing the human driver with a computer program. However, even though car companies are racing to bring self-driving cars to the markets, it will take decades until they become widespread. A more reachable solution would be through advanced driver-assistance systems (ADASs), in which we have seen lots of progress. Unfortunately, such life-saving technologies are expensive, and the masses cannot afford them. Especially considering that developed countries only account for 7% of road traffic death, not even ADASs will have a significant impact on road safety at global levels.

3.1 State-of-the-art

Many attempts are made to formalize *distracted driving*, but there is no consensus. Inattention is widely accepted as a major cause of unsafe driving and crash, Regan et al. [63] define driver inattention as "*insufficient or no attention to activities critical for safe driving*". According to the latest driver inattention taxonomy from the United States and EU Bilateral Intelligent Transportation Systems Task Force [64], drowsiness and distraction are two significant causes of inattention. According to Lee et al. [65]: "Driver distraction is a diversion of attention away from activities critical for safe driving toward a competing activity." Distraction affects driving in various ways. Dong et al. [66] categorize them into three groups:
1. Driver Behavior Patterns - This category covers patterns of actions such as rearview mirror checks or forward view inspection activities during driving. Drivers engaged in demanding cognitive tasks, lessen or some even abandon the visual monitoring of the mirrors or the instrument cluster Harbluk et al. [67]. More specifically, medium or heavy cognitive loads reduce the average area of the visual field to 92.2 and 84.41%, respectively [68]. These patterns are difficult to measure and, in most cases, require the use of intrusive sensors such as cameras.
2. Physiological Responses - Physiological indicators such as electroencephalogram (EEG), electrocardiogram (ECG) signals, skin conductance, and blinking rate fit this category. It is shown that EEG workload increased with working memory load and problem-solving tasks [69]. Researchers demonstrate that visual and cognitive distraction leads to a temperature increase on the skin surface [70]. The main issue with such metrics is the need for sophisticated sensors that are also uncomfortable or intrusive.
3. Driving Performance - Maintaining speed and lane-keeping are two examples of driving performance metrics that are affected by distractions. Zhou et al. [71] show that performing secondary tasks influence checking behavior (e.g., mirror checking) both in frequency and duration, which leads to a lower rate of lane changing. In this example, the decline in driving performance is a symptom of a change in *driver behavior patterns*— which falls in the first category. Liang and Lee [72] establish that cognitive distraction causes intermittent steering. Steering neglect and over-compensation are associated with visual distraction and under-compensation with cognitive distraction. Although sophisticated sensors are required to measure metrics such as headway distance—requiring

radar, available in cars equipped with adaptive cruise control (ACC)—or lane-changing behavior—requiring road facing cameras, available in cars equipped with LKA—such sensors are not intrusive towards the driver and are becoming more prevalent in the high-end vehicles.

The distracted driving literature has focused only on one or a hybrid of the categories mentioned above. However, most of the studies take one of the first two approaches, *driver behavior patterns* or *physiological responses*; we suspect because these approaches fit better in already established fields of medical and behavioral psychology research. Besides, such metrics are already well defined and perceived to be more reliable for distraction detection.

There is a large body of research that shows nomadic devices such as smartphones can be used for driver profiling and to detect unsafe driving. These works mainly use indicators such as harsh cornering or acceleration that can be obtained from smartphone sensors[1]. Only a few of these works focus on distraction detection. It is essential not to confuse unsafe driving with distracted driving, even though sometimes they manifest similar symptoms (such as harsh braking). Distracted driving is unsafe, but unsafe driving does not always constitute distracted driving. There are a number of ways that smartphones can be used to detect distraction. For instance, You et al. [74] use dual cameras on smartphones to detect drowsy and distracted driving. Bo et al. [75] use the magnetometer and the accelerometer from a smartphone to detect texting while driving. Lastly, Jiang et al. [76] show that wrist-worn smart devices can be used to detect distracted driving.

Next, we present examples of distracted driving detection methods that are relevant to our work. Wöllmer et al. [30] proposed to use head orientation to detect distractions, which mainly consists of user interactions with the instrument cluster. Additionally, they also use some of the vehicle operation signals such as pedal and steering wheel data as well as some driving performance metrics such as deviation from the center of the traffic lane and heading angle [30]. This rich set of signals were used as input to a long short-term memory (LSTM) recurrent neural network, which results in a subject-independent detection of distraction with up to 96.6% accuracy.

In a more relevant study, Jin et al. [77] develop two models based on support vector machine (SVM) called NLModel and NHModel, which are designed to detect low and high cognitive distractions, respectively. In this work, they only use data from the vehicle's internal network (CAN-bus) as input. Signals include vehicle operation parameters as well as vehicle dynamics. They achieve an accuracy of about 73.8% and a sensitivity of approximately 61.8%.

[1]For a survey of smartphone-based driver safety classification methods see [73].

In another study, Tango and Botta [31] propose a subject dependent model based on SVM that can achieve accuracy of up to 95%. They only use the dynamic signals of the vehicle for classification; however, data annotation is done accurately with the help of eye-tracking cameras and human supervision. The experiments are performed in a mostly straight highway drive and at the speeds of around $100\,\mathrm{km\,h^{-1}}$. Such over-simplifications raise questions regarding the robustness and reliability of the model in real-world environments.

Lastly, Öztürk and Erzin [78] propose the use of GMM for distraction detection. They extract cepstral features from the pressure on Gas and Brake pedals. Using GMM as a classifier and a decision window of length 360 s, they achieve 93.2% success to recognize non-distracted driving and 72.5% success in recognizing distracted driving [78].

In the next chapter, we propose a distracted driving detection mechanism that can detect distractions and warn the driver and, at the same time, be accessible to a considerable portion of car owners. Although this work is not the first attempt at solving this problem, it follows a novel and ambitious approach. Current literature mostly uses sophisticated sensors to detect distractions, sensors such as cameras for tracking head orientation [30] or measuring the skin temperature [70]. However, we propose to use the standard vehicle sensors; therefore, this method could apply to vehicles on the market.

Don't be satisfied with stories, how things have gone with others. Unfold your own myth.

Rumi

4 Non-Intrusive Distracted Driving Detection Based on Driving Data

In this chapter, we present a distracted driving detection approach that does not use any intrusive sensors, such as cameras or microphones; instead, we focus on driving data. We aim to classify driving segments as distracted or not distracted driving. Such a system can be used on-line to alert the driver in case of continuous distraction or off-line as a metric to judge the driving performance or asses riskiness.

First, we use a sliding window (frame) to extract features from driving signals, then using ML, we classify each window as distracted or not-distracted, then a decision function decides whether a sequence of frames represents distracted driving or not. We propose to validate this method using driving traces from 13 drivers chosen from a large driving Uyanik dataset [51] (see Chapter 2). For evaluation, we present the performance of the proposed method in terms of F_1 score and area under the receiver operating characteristic (ROC AUC). The remainder of the chapter is organized as follows, in Section 4.1, we describe the proposed methodology and the evaluation strategy. In Section 4.3, we present the evaluation results, and in Section 4.4, we discuss the obtained results. Lastly, in Section 4.5, we summarize and present future directions.

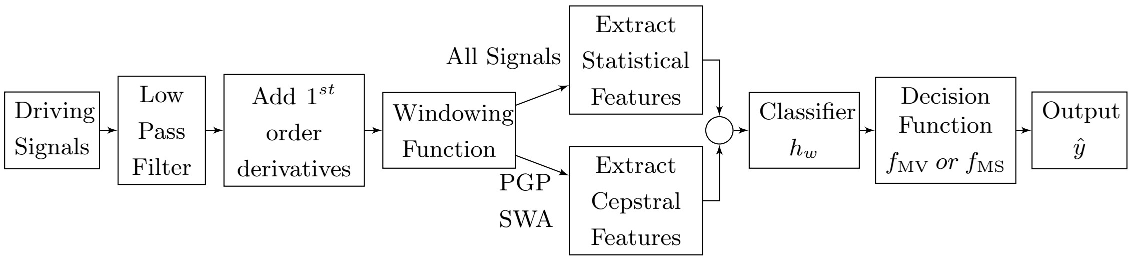

Driving Signals
Low Pass Filter
Add 1st order derivatives
Windowing Function
All Signals
PGP
SWA
Extract Statistical Features
Extract Cepstral Features
Classifier h_w
Decision Function f_{MV} *or* f_{MS}
Output $\hat{y}$

FIGURE 4.1: Distraction Detection Process

4.1 Methodology

We formulate the problem as a supervised learning problem. In which the input data is a multivariate time series recorded from a vehicle, and the target is 0 for normal driving and 1 for distracted driving. We denote a driving trace as $(x^{(i)}, y^{(i)})_{i=1}^{N}$, where x is the sensor measurements from the car, $y \in \{0, 1\}$ is the target label, and N indicates the total number of measurements. We seek a function h that predicts $\hat{y}$ for short driving sequences. Since x is a multivariate sequence, it is not possible to apply classic machine learning algorithms; therefore, we use the sliding window approach in order to apply conventional ML algorithms [79]. We use *window classifier* h_w to map each window frame of length w into individual predictions. Let $d = (w - 1)/2$ be the half-width length of the window, for a window frame at time t, h_w makes prediction $\hat{y}_t$ based on window frame $\langle x_{t-d}, \ldots, x_t, \ldots, x_{t+d} \rangle$. To reduce the computational complexity, we make window predictions for every $k = \lfloor w * (1 - \frac{r}{100}) \rfloor$ samples, where k denote *step size* and r indicates the percentage of overlap between two consecutive window frames. This results in $M = \frac{N}{k}$ window frames. Since it is possible that a window frame cover both distracted and non-distracted measurements, we label a window frame as distracted only if more than 50% of the measurements are distracted. Each driving trace $(x^{(i)}, y^{(i)})_{i=1}^{N}$ is converted into M window frames, then h_w is trained using feature vectors $\mathbf{x}$ computed for each window frame and its corresponding label. Similarly to classify an unseen driving trace x, it is first converted into window frames, then for each window frame at time t feature vector $\mathbf{x}_t$ is computed and h_w makes the prediction $\hat{y}_t$ based on $\mathbf{x}$. Lastly, we feed l consecutive window frame predictions $(\langle \hat{y}_{t-l}, \hat{y}_{t-l+1}, \ldots, \hat{y}_t \rangle)$ to the decision function f which produces the final prediction for the given sequence.

4.1.1 Evaluation Method

In order to utilize the entire driving traces $D = \{(x_i, y_i)\}_{i=1}^{K}$ for both training and testing, we employ a K fold cross-validation method called leave-one-group-out. Each time we train a model using data from $K - 1$ drivers and validate that on the data from the remaining driver. In other words, each driver is considered a group; therefore, at each fold, one driver is kept out of the training process, then h_w is trained and scored based on its prediction performance over the remaining slice. This is a subject-independent model because the model has no information about the driver that is being tested on. We evaluate the proposed mechanism using data from 13 drivers; therefore, in this case, $K = 13$.

Drivers typically drive attentively, but sometimes they get distracted by engaging in secondary tasks. We see the same pattern in our dataset; on average, only 36% of the measurements are labeled as distracted. Since the dataset is imbalanced in order to have a better measure of the proposed model's performance, we choose to report F_1 score as well as ROC AUC as the primary performance metrics. F_1 score is the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{1}{\frac{1}{recall} + \frac{1}{precision}} \tag{4.1}$$

Since our classification problem is binary, based on the application needs, we can modify the decision threshold. In order to avoid introducing new parameters, we employ the receiver operating characteristic (ROC), which demonstrates the diagnostic ability of a binary classifier as its discrimination threshold is varied. In fact, we use ROC AUC, which is a simple way of reporting an ROC plot using one numeric value (area under the ROC curve).

4.2 Video Annotation and Synchronization with Sensors

This section describes how we hand-annotate the distractions in a subset of the Uyanik dataset. It consists of two parts *a*) data annotation *b*) synchronization. The participants of the Uyanik data collection perform secondary tasks during the trip. By watching the videos from the driver-facing camera, we can mark the times the driver is engaged in these secondary tasks, which we use as a proxy for being distracted.

From the drivers with valid CAN and IMU data, we select the ones with available driver-facing video. We watched the videos and noticed that not all of them follow the same protocol. Several drivers skip some secondary tasks, or the tasks they perform are very different. This is particularly challenging because the conversations are in Turkish. Since we do not speak Turkish, based on their similarity, we had to decide if they follow the same protocol. The annotation consists of marking the beginning and end of each secondary task.

Having the timestamps of the secondary tasks on the video is not enough because the videos are not synchronized with the sensor data. To tackle the synchronization issue, we took inspiration from a work by Fridman et al. [80]. The general idea is that in a moving vehicle, turning the steering wheel results in lateral movements of the car. These lateral movements are visible in the front-facing camera feed, and they should correlate with steering wheel movements. Therefore if we find the time lag that maximizes the

Table 4.1: Selected Signals for Driver Distraction Detection

Sensor	1st order Derivative	Cepstral
Percentage Gas Pedal (PGP)	Yes	Yes
Str. Wheel Angle (SWA)	No	Yes
Str. Wheel Rel. Speed (SWRS)	No	No
Vehicle Speed (VS)	Yes	No
Engine RPM (ERPM)	Yes	No
Pitch	No	No
Roll	No	No
Yaw	No	No

cross-correlation between the two signals, we can use it to synchronize the videos with the other sensor data. To achieve this goal, first, we use the front-facing video feed to estimate the vehicle's lateral movements. A dense optical flow method based on Gunner Farneback's algorithm [81] is then applied to the video. The result is a two-channel array representing the optical flow vectors. These vectors point to the direction of the movement of image pixels. We take the average of the horizontal component of all pixels and maximize its cross-correlation with the SWA. This method produces satisfying results for the majority of the recordings. In this work, we annotate and synchronize data from 13 drivers, composed of 2 female and 11 male drivers.

4.2.1 Feature Extraction

In order to remove high-frequency noise, we apply a low pass filter to all the signals to smoothen the signals. Then for some of the signals (Shown in Table 4.1) we derive the temporal derivative of the signals. For both smoothing and computation of derivations, we use an implementation of the Savitzky-Golay algorithm [82]. Figure 4.1 shows the various stages that the signals go through before the feature extraction step. In this study, we use eight signals, which are listed in Table 4.1. These signals are obtained from CAN-bus or IMU. Then we apply a windowing function to all the signals, which takes two parameters, w to determine the length of the frame and r to specify the amount of overlap between the two consecutive frames. This will break down the signal into frames of size w and is ready for the feature extraction stage. Per each signal, nine statistical features are extracted from each frame. These descriptive statistics are selected to be representative of the distribution of sensor values covered by the frame. This set includes minimum, maximum, mean, median, standard deviation, kurtosis, skewness, and the number of zero crossings. We also extract cepstral features from two signals, PGP and SWA, these are the only vehicle operating inputs that are directly operated by the driver. Earlier studies have shown that cepstral analysis is suitable for driver identification [35, 54], we suspect that they are also useful for detecting distracted driving.

The implementation is as follows; we multiply a Hamming window of length w to limit the signal and also reduce the edge effects. We denote $c(k)$ to be the first k cepstral coefficients, and higher-order coefficients are discarded. The equation to derive $c(k)$ is:

$$c(k) = \left|\mathcal{F}^{-1}\left\{\log(|\mathcal{F}\left\{x_t w_t\right\}|)\right\}\right| \tag{4.2}$$

Where x_t denotes signal values covered by the current frame, w_t denote Hamming window with the same length as the frame and $\mathcal{F}$, and $\mathcal{F}^{-1}$ denote discrete Fourier transform (DFT) and inverse-DFT (IDFT) respectively. We extract cepstral features from two signals, PGP and SWA, and keep the first $k = 32$ coefficients as cepstral features.

Table 4.2: Average Scores For Each Signal

Signal	Mutual Info.	F-Test
SWRS	0.517	0.084
ERPM Derivative	0.510	0.059
Pitch	0.490	0.056
PGP Derivative	0.475	0.035
Roll	0.470	0.012
PGP	0.436	0.060
ERPM	0.428	0.359
VS Derivative	0.427	0.088
Yaw	0.425	0.054
VS	0.415	0.792
SWA	0.228	0.061
Cepstral Coefficients PGP	0.029	0.036
Cepstral Coefficients SWA	0.025	0.026

4.2.2 Feature Importance

We analyze the extracted features to evaluate their fitness for our experiments. We use two metrics for this purpose, the conventional ANOVA F-test and mutual information (MI) [83, 84] as two metrics to evaluate the importance of the features as well as some insights into which features or signals are more important for our classification task. To get an idea that which signal is more important, first we compute the MI for all features, we normalize it and compute the average importance for five best features from each signal. Since we compute nine features for each signal, it is unrealistic to expect all the features from a signal to be equally informative. Therefore we choose a number of the best features per signal (in this case, 5) as a proxy for the importance of the signal. The corresponding results are presented in Table 4.2. Since the methods based on F-test only estimate the linear dependencies while MI captures linear and non-linear dependencies, we sort the signals in descending order of their MI score. Among the signals, we can observe that features extracted from SWRS have the highest MI score, and the cepstral features are the worst.

We perform a similar analysis to assess which functions used for feature extraction are the most informative. The corresponding results are presented in Table 4.3. We can

Table 4.3: Average Scores For Each Function

Function	Mutual Info.	F-Test
Min	0.955	0.310
Max	0.933	0.289
Range	0.495	0.076
Median	0.136	0.309
Mean	0.092	0.314
Standard Deviation	0.053	0.067
# Zero crossings	0.040	0.166
Cepstral Coefficients	0.034	0.039
Skewness	0.032	0.041
Kurtosis	0.030	0.081

see that simple functions such as Min, Max, Range, Mean are more useful that Skewness and Kurtosis.

4.2.3 Classification Algorithms

We select five classification algorithms to be used as *frame classifier* h_w. The following is the list of classifiers used in this study along with their corresponding parameters.

AdaBoost (AB). AdaBoost, as introduced by Freund and Schapire [85], suggests the use of a collection of weak learners. As the number of weak learners increases, the algorithm increases the weight for examples that are more difficult (are currently misclassified), therefore improving the performance. We use the implementation from the scikit-learn library [86], which uses the AdaBoost-SAMME algorithm [87]. We use a decision tree with a maximum depth of 1 as weak learner [88]. Two hundred decision trees as a weak learners, learning rate = 0.75.

Gradient Boosting (GB). Gradient boosting, iteratively combines weak learners into a single strong learner. At each iteration, GB fits a tree to correct the mistakes of the model from previous iteration. Therefore in principle, Gradient boosting fits models to the residuals of the previous iteration. These residuals are equivalent to the negative of the gradients [89]. 100 estimators, maximum depth = 6, maximum features = None, maximum depth = 6, learning rate = 0.05.

Random Forest (RF). Random Forests were introduced by Breiman [90]. As the name suggests, it consists of a combination of tree predictors. Each tree is trained on a subsample of the examples, which is also known as bagging. Moreover, to further reduce the correlation between the trees and avoid overfitting, each tree is fitted on a subset of features. We use the implementation from scikit-learn package [86]. With parameters: 200 estimators, maximum features = None, maximum depth = 7, class weight = balanced.

Support Vector Machine (SVM). Lastly, we step away from tree-based algorithms and attend to once upon a time king of classifiers, SVM. SVMs for some time were known to be the best classifiers available. Vladimir Vapnik introduced the linear support-vector machines in 1965. SVMs are binary classifiers, in linear case SVM finds the maximum-margin hyperplane that separates the two class. Cortes and Vapnik [91] in 1995 proposed to use the kernel trick [92] and extended SVM to solve non-linear classification problems. In this approach instead of dot products in SVM a non-linear kernel function is used. Therefore the algorithm is able to fit maximum-margin hyperplane in a transformed space, which allows for separation of non-linear feature boundaries. We use the SVC classifier from scikit-lean library [86] which uses LIBSVM library [93]. RBF kernel, C = 0.1, $\gamma = 0.01$, class weight = balanced.

k-Nearest Neighbors (k-NN) k-nearest neighbors (k-NN) in order to classify an unlabeled example finds the k closest examples in the training set and selects the majority class. # neighbors = 5.

In our experiments, we use the implementations from the scikit-learn software package. All other parameters are set to their default values as of scikit-learn version 0.19.2 [86].

4.2.4 Decision Functions

The decision function f determines the final prediction $\hat{y}$ at time t based on the last l frame predictions, we call this a decision window: $\langle \hat{y}_{t-l}, \ldots, \hat{y}_{t-1}, \hat{y}_t \rangle$, therefore l is the number of frames covered by a decision window. Below we introduce two decision functions to obtain $\hat{y}$ and evaluate their performance later in Section 4.3:

1) Majority vote (MV) Let d to denote count of the frames in the decision window that are predicted as distracted driving. Then a decision window is classified as distracted if $\frac{d}{l} > 0.5$.

2) Maximum score (MS) Let d' to be the cumulative classification score for the distracted class. Then a decision window is classified as distracted if $\frac{d'}{l} > 0.5$.

4.3 Model Performance

In this section, we present the results of our experiments. In each subsection, we focus on one of the components of our proposed methodology and discuss its implications.

4.3.1 Frame-size Analysis

In order to find out the optimal frame-size, we try several sizes[1] as well as overlap ratios[2]. Since we do not want to have results influenced by choice and working mechanism of the decision function, in these experiments, we do not apply the decision function f and only consider the predictions from h_w. The results are presented in Figure 4.2, the reported numbers are the average cross-validated scores for each combination of r and w

[1]Frame-sizes of 4, 7, 10, 15, 20, 30, 45 and 60

[2]Overlap ratios of 0, 25, 50, 75 and 90 percent.

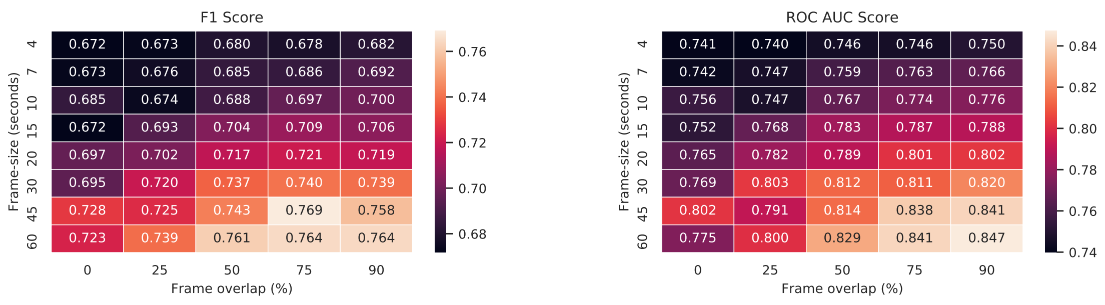


FIGURE 4.2: The effect of frame-size on classification score. Both F_1 and ROC AUC follow a similar pattern. Increase in frame-size and overlap ratio improves the score.

parameters. The results improve as overlap r increases, this can be explained by the fact that an increase in overlap also increases the number of frames and therefore having more examples for the classifier to learn from. It is also evident that performance improves as the frame-size w increases since our features are mostly descriptive statistics, having larger frames filters out noise from the features; therefore, larger frames yield better results. From this section, we can conclude that for both parameters r and w, it is best to choose a larger value, however such choices have some drawbacks as well. Larger values for r lead to an increase in the number of examples, and feature calculations, therefore, becomes computationally more expensive. On the other hand, larger w results on delays in the predictor, therefore w and r should be selected in a manner suitable for application needs.

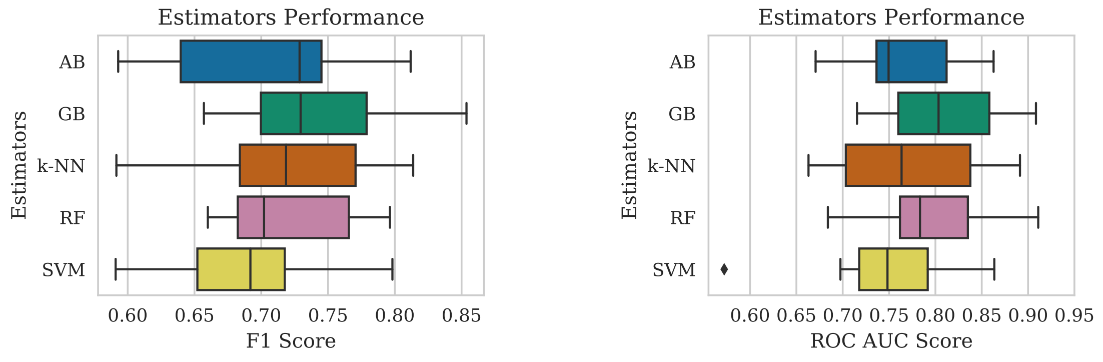


FIGURE 4.3: Estimator Performance Comparison - $w = 30s$ and $r = 75\%$

4.3.2 Classifier Benchmarks

We run our experiments for all of the 5 selected classifiers and compare their performance as h_w, meaning we only score the classifiers for their ability to predict individual frames. The results are presented in Figure 4.3 (obtained using $w = 30s$ and $r = 75\%$), when considering ROC AUC score GB yields the best performance and RF takes the second place.

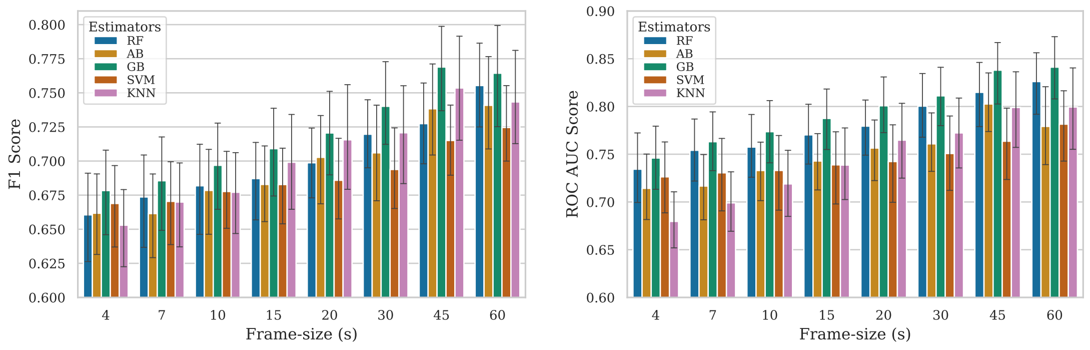


FIGURE 4.4: Estimators performance comparison for several frame-sizes - $r = 75\%$. Error-bars correspond to 95% confidence interval.

TABLE 4.4: Decision Function Performance - Results are for k-NN as classifier

Frame	Decision Window (DW)		Closest Frame-size		Decision Function	
Size (s)	DW Len.	DW Duration (s)	Len. (s)	ROC AUC	ROC AUC MS	ROC AUC MV
4	5	9.00	10	0.718	0.748 (+4.07)	0.718 (-0.09)
4	10	14.00	15	0.738	0.780 (+5.61)	0.759 (+2.80)
4	15	19.00	20	0.764	0.799 (+4.60)	0.782 (+2.34)
7	5	15.75	15	0.738	0.774 (+4.84)	0.751 (+1.74)
7	10	24.50	20	0.764	0.812 (+6.28)	0.799 (+4.57)
7	15	33.25	30	0.772	0.836 (+8.29)	0.825 (+6.92)
10	5	22.50	20	0.764	0.801 (+4.83)	0.760 (-0.58)
10	10	35.00	30	0.772	0.844 (+9.38)	0.814 (+5.44)
10	15	47.50	45	0.798	0.871 (+9.05)	0.846 (+5.89)
15	5	33.75	30	0.772	0.822 (+6.48)	0.802 (+3.89)
15	10	52.50	45	0.798	0.867 (+8.59)	0.855 (+7.09)
20	5	45.00	45	0.798	0.855 (+7.12)	0.822 (+2.96)

We also investigate how different classifiers exhibit different behaviors due to the changes of frame-size w. Figure 4.4 depicts the average cross validated ROC AUC and F_1 scores for various frame-sizes and fixed overlap ratio of $r = 75\%$. For example, k-NN clearly benefits from a larger frame, because as it was discussed larger frames result in smoother features. On the contrary SVM does not improve as much as other classifiers do, this may improve by tuning the hyperparameters for every frame-size however it is out of scope of this work.

4.3.3 Decision Function Benchmarks

Decision function f can be seen as a meta classifier which simply aggregates predictions from l consecutive frames and outputs a single prediction. We investigate its performance by running experiments using values of 5, 10 and 15 for l and similar combinations for w as previous experiments. To compare the two proposed decision function a small sample of results (Cases with DW Duration $< 60s$) is presented in Table 4.4. The column DW Duration, in the table refers to the time-span covered by the decision function f to make the prediction, these values are computed considering overlap ratio of 75% between the consecutive frames.

In order to demonstrate the advantage of using the decision function, we also present the scores for individual frame predictions, to have a fair comparison for each row we compare the decision function with the closest frame-size. For example, if $w = 4s$ and $DWLength = 15$, DW Duration is 19 seconds, meaning the decision function effectively uses data from the past 19 s to make a prediction. We compare this instance with the prediction results for h_w (classifier without decision function) with the $w = 20s$, because it is the closest frame-size to 19s that we have considered. In the table we can see that in f_{MS} results in between 4 to 9.3% (6.6% in average) improvements over not using a decision function. f_{MV} performs worse than f_{MS} with average improvement of 3.58%.

4.4 Discussion

First of all, we would like to mention a limitation of the proposed approach. If we consider how the data-set is annotated and the signals we use for classification it is not expected to be able to achieve excellent results for detecting distractions. This is because we mark a long stretch of driving as distracted driving, however although in that period driver is engaged in a certain secondary task, this engagement is not uniformly present throughout the stretch, therefore we are inadvertently injecting noise into our training data. In fact one could perform the labeling more granularly and potentially improve the results.

On the other hand, the proposed methodology is quite flexible and can be optimized for the intended applications, generally one needs to find the right balance between the quality of detection and delay in detection. For example, using GB classifier, $w = 60$, $r = 0.75$ and $DW = 15$ (DW Duration of 285s) we achieve ROC AUC of 98.7%. If we need a shorter the detection time we get to ROC AUC of 92.7% with k-NN classifier, $w = 20$, $r = 0.75$ and $DW = 15$ (DW Duration of 95s). For DW durations less than 60 seconds the results are presented on Table 4.4, for smaller DW durations, ROC AUC decreases.

We should also point out that our evaluations are in a subject-independent manner because the model is always trained on drivers that are not among the test set. This indicates that distracted behavior is some degree similar among drivers. Tango and Botta [31] indicate that subject-specific training—training a model for each driver—resulted in about 20% improvement in the performance; however, since that approach is unrealistic it was not explored in this work.

Moreover, we do not use any intrusive signals, not tracking the driver's head orientation nor the cabin sound, only sensor data from the car. Not only that, studies such as [30] and [31] regardless of the intrusive sensors, they do use metrics that are not readily available—such as distance from the center of the lane—and therefore limit the deployability. However, our goal is to make a similar prediction without having access to such information. With the current setup, we may not be able to address applications that require accurate spontaneous and momentarily distraction detection. Instead, the system performs well at detecting long-lasting distractions, such as mobile-phone or in-car conversations. Moreover, one can use such a system to characterize risky driving. It is crucial for insurance, logistic, and public transport companies to keep track of their customers' or employees' risky driving behavior. Such application is equally beneficial for individuals who would like to keep track of their driving quality. For example, parents who are concerned about the safety of their teen, would like to know whether or not their child is a risky driver.

The results show that the performance is improving as the overlap and frame-size increases. We perform experiments with a fixed set of overlaps for a fixed set of frame-sizes. The cost of having a large overlap is a high computational cost; however, as the frame-size increases, the frequency of calculation is significantly reduced. Therefore, to improve the results, one could use larger frame overlaps. For instance, with a frame-size of 30 seconds, 75% overlap amounts to 7.5 seconds stride. Therefore, increasing the overlap to 90% should not be an issue, although it leads to more frequent computations at the small frame-sizes.

To achieve a practical solution, we need to reduce the false-positives. The dataset we used is small (13 drivers). A larger, precisely annotated dataset is needed to improve the results. With enough time and resources, it is possible to use ML/DL approaches to automate the annotation process, and annotate the rest of the Uyanik dataset. Moreover, we focused on the subject-independent models because it is unrealistic to have access to annotated data from the driver. An idea worth exploring is to adapt a subject-independent model to the user's driving style.

4.5 Summary

In this chapter, we have proposed a mechanism to detect distracted driving based on non-intrusive vehicle sensor data. In the proposed method, eight different driving signals are used. The data is collected, two types of statistical and cepstral features are extracted in a sliding window process, next a classifier makes a prediction for each frame, and lastly, a decision function takes the last l predictions and makes the final prediction for the given frames. We have evaluated the subject-independent performance of the proposed mechanism using a driving data-set consisting of 13 drivers. We analyzed the implications of changing the size of the sliding window and its overlap ratio. We have shown that the performance increases as the frame-size and decision window become larger. We compared the performance of several classifiers. The best results were obtained using GB classifier $w = 60$, $r = 0.75$, and $DW = 15$ (DW duration of 285 s), which yields ROC AUC of 98.7%. We showed that it is possible to detect long-term distractions using driving data. Such a solution can provide timely feedback to drivers and encourage them to drive safely, therefore reduce the traffic-related deaths. Moreover, since the proposed method uses minimal sensors and is trained independently of the driver, it is suitable for wide deployment.

Part II

Driver Identification

If someone steals your password, you can change it. But if someone steals your thumbprint, you can't get a new thumb. The failure modes are very different.

Bruce Schneier

5 Introduction to Driver Identification

In this chapter, we clarify what what do we consider as driver ID. Then, we look into some of the applications of driver ID. After that, we explore what factors affect the design of a driver ID system. We review the state-of-the-art research and identify some key points to address in this dissertation.

What is a driver ID system? When it comes to identifying individuals, biometrics identifiers are the first to come to mind. These are systems that employ personal traits, such as fingerprints, retina scans, voice, or facial features for identification. Biometric methods are mature technologies, but they come with some limitations. As an example, a facial recognition system requires the installation of a camera pointing to the driver, which raises privacy concerns. Similarly, fingerprint scanners have proved to be susceptible to copy [94, 95].

The research community has proposed several non-biometric solutions. For example, Riener and Ferscha [96] propose to install sensors inside the driver seat to ID drivers based on their posture and body structure, apart from the tremendous costs, drivers find these sensors intrusive. Even if we accept the risks and discomfort, the wide deployment of such systems is costly and cumbersome to maintain.

We define driver ID as the task of identifying a particular driver from a group of known drivers based on their driving behaviors. We exclude conventional biometric methods such as fingerprint or retina scans. Moreover, we are not interested in face recognition as well as trivial radio frequency identification (RFID) tags, or smart-card based solutions. Additionally, we aim to fill the gap where such conventional methods

Table 5.1: Driving data sources and their merits

Data source	**Common sources**	**Pros**	**Cons**
CAN-data High sample-rate	CAN-bus Black-boxes Dongles	High utility High processing load & storage Wide range of vehicle sensors	Requires an external device Or provided by the manufacturer Vehicle dependent
FCD Low sample-rate	Smartphone apps Car API	Easy to collect Vehicle agnostic Contextual information such as road profile can be obtained from location trajectories	Lower reliability No access to vehicle sensors

are inadequate. We can use driver ID in conjunction with the traditional identification methods, in order to verify the claimed identity or trigger a more reliable identification method when necessary. Therefore we think the focus should be on driver ID based on driver behavioral features.

Driving is a complex task that manifests various indicators such as route choice, lane preferences, braking, and acceleration patterns. Some features are direct driver input, such as pedal operation or steering, some are inferred data, like vehicle dynamics, such as acceleration patterns or speed profiles. Driver-related differences are the result of behavioral, learned driving habits, and even anatomical features of drivers. We can use these differences to identify the driver. These metrics must be sufficiently stable, meaning they should not change daily and stay consistent for months and even years.

Driver ID methods in the literature rely on two categories of data: 1) *CAN-data*, often with high sample-rate, and a large number of features. This kind of data is usually collected from CAN-bus, using OBD-dongles, black-boxes or external sensors. 2) *FCD* provided by GNSSs. Usually lower sample-rate and noisy data which can be provided by smartphones, car-navigation system or external receivers. Automotive manufacturers have access to both these data categories, and some even provide APIs to third-parties to access such data. Although original equipment manufacturers (OEMs) can use the high sample-rate data internally, because of data transmission and storage costs, third-parties can only access low sample-rate data.

In the near future, third-party service providers may get access to CAN-data for applications that are deployed on the infotainment systems, perhaps through Android Auto or Apple CarPlay. As an example, Daimler has recently launched a platform going into this direction[1]. Without manufacturers' support access to the CAN-data requires installation of external devices to collect data from the onboard diagnostic (OBD)-II port or set up external sensors. Such a solution is vehicle specific which adds complexity to the system. On the other hand, FCD is more convenient to obtain, either through

[1]`https://developer.mercedes-benz.com/apis`

an externally mounted receiver or a smartphone application (less reliable). Table 5.1 summarizes the pros and cons of each data category.

We can identify the drivers in several ways. We propose three category of features that can be used to characterize human driving:

1. **Spatial features**, refers to the information that we can extract based on knowing the geospatial location of the vehicle, origins and destinations and the routes taken by the driver.

2. **Temporal features**, refers to temporal aspects of a trip, that is, the hours of the day or day of the week. Imagine a working mother who commutes to work every morning at 7 o'clock, and returns home at around 6 o'clock. Now consider her spouse, who stays home to take care of the baby and only leaves the house for occasional grocery shopping. It is easy to distinguish between the two only based on the hours of driving.

3. **Behavioral features**, we consider a feature behavioral when it does not fit in any of the above categories, such as acceleration patterns, pedal actuation, steering actions, choice of speed and over-speeding, overtaking habits.

5.1 Applications of Driver Identification

Here we present an overview of the potential applications of the driver ID to motivate exploring this subject. This list is by no means exhaustive, instead it is a selection of applications.

Fleet Monitoring

Ride-sharing apps such as Uber, Lyft, Bolt, have revolutionized transportation by providing a level of comfort that once deemed unimaginable, but they have issues of their own; There have been many reports of sexual assault and various other crimes by ride-sharing drivers [18, 97], in addition there are cases that drivers commit fraud or cheat the system. They create fake GPS traces, create ride requests from stolen accounts and more [19]. Introduction of background checks by the ride-sharing companies, which is now mandatory in some jurisdictions, has reduced the crime rates, but new workarounds emerge. In such cases, your Uber driver may not be who you think they are. Uber has a system in place that sometimes asks the driver to stop and authenticate themselves using a selfie picture [98]. It is unclear how often or under what criteria it is triggered. A similar challenge exists for on-demand delivery services such as Deliveroo, and even traditional fleets. Using driver identification your ride-sharing company could verify driver's identity based on their driving style.

Sleepiness/fatigue increases the likelihood that a commercial vehicle involved in a collision be fatal [12]. Regulations are in place to prevent long-haul drivers from overwork. Meanwhile there are indications that drivers under the pressure of their employees or on a voluntarily basis, disregard the regulations regarding the working hours [15, 16]. In Europe, instruments such as smart tachographs (directive 165/2014 [14]) were introduced to enforce adequate rest periods. However there drivers tamper with these tachographs [17]. Since overworked and sleep deprived drivers have much higher risk of crash involvement [8–11] they also pose a risk to other road users. Driver ID using driver behavior profiling can be a solution to this problem. Any company owning a fleet of vehicles would benefit from a friction less way of verifying driver attribution to vehicles.

Usage-Based Insurance

Usage-based insurance (UBI) had a market size of 35 billion $ in 2017 and is estimated to grow to 107 billion by 2024. UBI is also known as Pay-As-You-Drive (PAYD) or Pay-How-You-Driver (PHYD), with the former emphasizing on charging based on the mileage and the latter to focus on driving riskiness. A few examples of UBI services are Allstate Drivewise, Aviva Drive, MetroMile, Progressive's Snapshot, and Nationwide SmartRide. UBI is enabled through various ways, OBD-II, smartphone, blackbox and embedded telematics, which means that UBI providers have an infrastructure already in place that can be used potentially for driver ID. Driver ID can enable novel products in this space. For example, with driver ID, multiple drivers, sharing a vehicle can each have their insurance policy. For instance, in the case of a family with a teen driver, insurance companies can charge a higher premium for the teen and lower premiums for the parents (assuming that the parents are safe drivers). We can extend this to shared company cars.

Comfort Features

Another use-case arises in modern vehicles, where the number of comfort-related features has increased substantially. Being able to identify different drivers can enable the automation of comfort settings (e.g., seats and mirrors position, air-conditioning, adaptive driving-assistance system profiles). Therefore when a car is shared between a group of drivers, when one starts to drive, the car detects the driver and applies the customizations to that driver's liking.

Suppose, a car that could intelligently adapt—in order to optimize the energy efficiency of the engine—to the driver's driving style. The first time this process may require a few hours of driving data to construct a profile. Using driver ID, when a new driver sits behind the wheel, the car can recognize the new driver, discard or shelf the old profile, and fit a new profile to the current driver's driving style. Therefore, adaptively optimize energy consumption and the driving experience.

Application Scenarios

To further solidify applications of driver ID, let us look at a few concrete scenarios:

- Bob purchases subscription to a service provided by the Acme company. This service can detect stolen cars and inform the Police. Acme can get access to the car data through the manufacturer's infrastructure. A few months later, Bob receives a text message from Acme that they suspect his car is stolen and they ask his confirmation to inform the police. Bob remembers that he lent his car to Carol without indicating it in the Acme's app. Therefore he responds to Acme that the car is not stolen, and there is no need to inform the authorities.
- A speed-trap captures a photo of a car speeding way above the limit, but the driver's face is not visible. Alice claims that she had borrowed her car to Bob. Since the car is equipped with an insurance dongle, the Police, with the collaboration of the insurance company, can get access to the data already uploaded to the insurance company's server. Using that data, they can confirm that Bob was driving at the time.
- Alice and Carol rent a car, but since they do not want to pay extra for the second driver they only specify Alice as the driver. After a few hours of driving, they stop at a gas station, since Alice is already tired, Carol sits behind the wheel and drives the car for the rest of the trip. Since the car rental company has equipped all their vehicles with blackboxes when Alice and Carol return the car, the rental company charges them with the fee for the second driver.
- Alice buys a new low-cost auto insurance policy, which only permits her to be the driver. She considers herself to be a safe driver, so she also installs the insurance company's dongle in her car that allows her to qualify for further discounts. One day she lends her car to a friend to run some errands. A few hours later, the insurance company finds out about the unregistered driver and cancels Alice's insurance.

5.2 Driver Identification as a Privacy Issue

We introduced driver ID as a function that can be provided by third-party service providers or car manufacturers. However, we can look at it from a different perspective. Suppose that a family of two who share the same car, allow a third party company to collect floating car data, in order to provide suggestions as to when and where fill

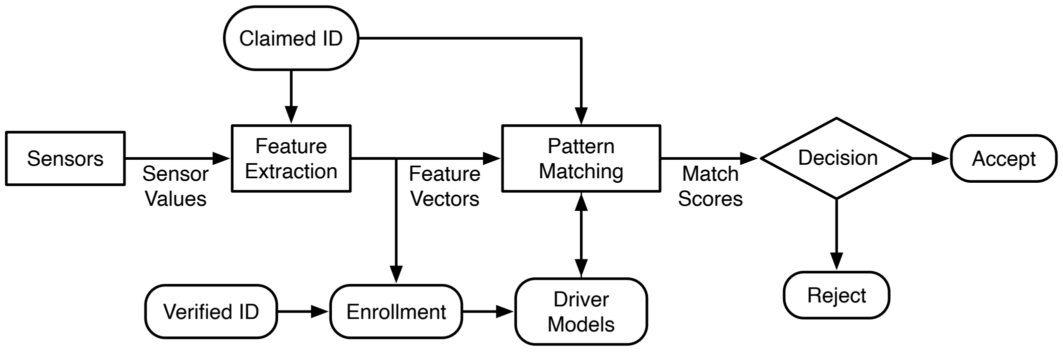


Figure 5.1: Generic driver verification system.

up the tank to minimize their costs. This company could perform driver identification without the family's knowledge in order to, for example, send targeted advertisements according to the family member driving the car. The security and privacy research community would refer to this task as driver fingerprinting. To construct a fingerprint for each driver that allows an adversary to identify (fingerprint) the driver in the future.

5.2.1 Threat Model

To continue further with security terminologies here, we establish the threat model corresponding to the problems we will consider in this work. We assume that the adversary has access to the internal vehicle network (e.g., CAN bus) or obtains the dataset of recordings previously uploaded to the server, this could be done using car's telematics unit or external data collection devices. Moreover, the adversary is a passive eavesdropper. Therefore there will not be any interaction between the adversary and the car. Because it has become more commonplace for car data to be shared and stored, this is not far fetched. Although we assume and hope that companies are honest and abide by the privacy regulations such as the GDPR. There are situations when a data breach occurs or a company is subpoenaed to hand over the data to the authorities, therefore even though data collectors may not wish to play an adversarial role, they may enable others to achieve adversarial goals.

5.3 Elements of a Driver Identification System

Driver verification and identification are closely related. First, we introduce the major components of a driver verification system. Then we explain how similar components can be used to construct a driver ID system. The general approach to driver verification consists of 5 steps: *a*) sensor data acquisition *b*) feature extraction *c*) pattern matching *d*) decision making, and *e*) enrollment. A block diagram of such a system is shown in Figure 5.1. The system has two phases, enrollment and prediction. In the enrollment phase, the feature extraction stage maps a driving interval to a multidimensional feature vector. We seek features that exhibit high driver discrimination power, high inter-driver variability, and low intra-driver variability. This feature vector is used to construct a reference model λ for the driver, the details of this reference model are closely related to the pattern matching algorithm used in the system. In the prediction phase, the feature extraction stage maps a driving interval to a feature vector, similar to the enrollment phase. The feature vector $\mathbf{x}_i$ is compared or scored by the pattern matching algorithm. This results in a match score z_i for each feature vector i. The match score measures the similarity of the input feature vectors to the reference model or pattern of the claimed driver. Finally, a decision is made to either accept or reject the claimant according to the match score or sequence of the match scores, which is a hypothesis-testing problem.

In a driver ID system, capable of identifying K drivers, at the enrollment phase, K reference models are constructed, one for every driver. At prediction time, the pattern matching algorithm outputs match scores z_i^j for feature vector i and driver j. Lastly, the decision-making stage, instead of the binary choice of accepting or reject, should select the most likely driver. We can also design the system in a manner that, in case of low confidence, the system selects the reject option. The reject option signals the lack of confidence, or that the feature vector is not likely to belong to the drivers enrolled in the system.

5.4 Design Constraints

Several factors affect the design of a driver ID system. We point out a few of the most important issues.

5.4.1 Communication and Computation

Whether we can rely on communications between the car and a central server will affect the design of the system. If no communication is possible or if the cost of downloading data from the server is high, perhaps the ideal situation is to download the model to the car, and the whole process must be done locally. Other considerations may be required to see what is the best approach for training the model, can it be done locally? Is it possible to extract features locally and send the compressed features to the server for training? The amount of computational power available locally (in-car) limits the choice of the algorithm and the model used in a driver ID system.

5.4.2 Data Availability

A more extensive set of signals and more frequent generally leads to a more accurate and faster driver ID system. The use of *spatial features* demands location data. For *behavioral features*, we need to collect high-frequency measurements from the vehicle, which is often obtained from the vehicle's internal network (e.g., CAN-bus) or by installing blackboxes. Such efforts will incur costs and raise privacy concerns that need to be addressed. Therefore, if we only have access to is 1 Hz FCD. We cannot use the most accurate driver ID methods that require high sample-rate pedal operation data. Instead, we should resort to a solution only using FCD.

5.4.3 Use-case Requirements

Depending on the use-case, some of the features are susceptible to errors. For example, to detect a stolen car, one cannot just rely on temporal or spatial features. It is possible the genuine driver in the morning, instead of going to the office, drives to the other side of the city for a dentist appointment, in this case, spatial features will lead to a false negative. While in another case, the car may be stolen and driven away at the same hour as the driver regularly would commute to work, again, our model would end up with a false negative. Therefore one must be mindful of the features used and what effects they may have on the driver ID system.

5.5 State-of-the-art

In this section, first we give an overview of the methods used in the literature. Then we briefly present some of the more notable works, first those using CAN-data and then those that use FCD. Table 5.2 presents an overview of the driver ID literature, in terms of the methodological approach, and results.

5.5.1 Taxonomy of Related Work

Early works on driver ID, suggest using car following models such as optimal velocity (OV) and Helly [99]. Since the results were not promising, the focus was shifted towards statistical models of driving signals. In terms of the models, several works use GMM motivated by its success in fields of speech/speaker recognition [35, 36, 56, 78, 99]. Some authors use traditional artificial neural networks (ANNs), namely multi-layer Perceptron (MLP) for driver identification [37, 42, 100, 101]. The extreme learning machine (ELM) [102], an ANN method closely related to MLP, is used in [38, 39]. Fuzzy ANNs, in particular, adaptive network-based fuzzy inference system (ANFIS) is proposed in [100]. SVM [91] that in the early 2000s was considered to be the state-of-the-art classification method has its fair share in the driver ID literature [40, 41, 46, 101, 103–108].

Commonly, researchers experiment with a wide variety of classifiers, but we only mention the best performing or popular methods. In the past few years ensemble methods, especially tree-based boosting and bagging methods are used quite often, for instance, random forest (RF) is used in [40, 42, 43, 47, 49, 101, 106, 107, 109, 110], or some works used other variations of tree-based algorithms [42, 43, 46, 101, 107, 111]. Among the simpler algorithms, we can point to k-NN, used in [40, 42, 43, 46, 105, 107] and naïve Bayes (NB), used in [40, 46]. Other approaches in the literature are linear discriminant analysis (LDA), proposed in [112], some works propose to use an outlier detection or rejection method [41, 108]. The use of dynamical models, such as hidden Markov models (HMMs) is explored in [113, 114]. More recently, some authors suggest DL-based methods [44, 45, 48, 50, 115].

A wide variety of data sources is used in the literature. The majority of research uses data collected from the CAN-bus or OBD-II [38–45, 47, 49, 50, 56, 99, 104–106, 109, 111, 112, 116]. Some works use driving simulators or video games to collect data [35, 99, 103, 107, 113, 114]. Several works use the data from pressure sensors retrofitted under the pedals in conjunction with CAN-bus data [35–38, 78, 99, 100, 117]. Some authors use laser rangefinders that measure the distance to the leading vehicle [38, 39, 78]. IMU is another data source that is commonly used for driver identification [38, 39, 46, 48, 101, 108]. Some works use GNSS receivers often retrofitted or integrated with in-vehicle data recorders (IVDRs) [46–48, 108, 112], and some enrich the data using map data [48]. In recent years, as smartphones become ubiquitous and more powerful, more attention is paid to the potential use of smartphones for driver ID [43, 104, 110, 115, 118, 119].

In terms of the features, many works use the raw (unmodified) signals, but a more conventional is the use of statistical features. Additionally, some researchers proposed more complex features; for example, cepstral features that are common in speech processing were first proposed by Miyajima et al. [35] and adapted by other researchers [36–39, 56, 78, 104]. Spectral features—often known as fast Fourier transform (FFT) based features—are also shown to be effective [38–40, 56, 103, 104, 113]. Discrete wavelet transform (DWT) is used by [47, 100]. Some works use event-based features. They extract features for specific events (e.g., a turn), or count the number of events (e.g., braking) [41, 47, 48, 50, 112]. Other researchers considered the origin, destination or points of interest (POIs) [46], or the date and time [46].

5.5.2 Most Relevant Studies

Here we present a summary of some of the more significant works using CAN-data. Wakita et al. [122] propose to identify drivers using behavioral signals captured during the car-following task. They compare the identification performance of GMM against physical models. In terms of features, they use the following distance (FD), VS, brake, and gas pedal pressures. They show that GMM performs better than physical models. In a follow-up work, Miyajima et al. [35] propose to use features based on cepstral analysis of gas and brake pedals. Cepstral features, especially Mel-frequency cepstral

TABLE 5.2: Summary of Driver Identification Literature

Reference Author/Year	**Data-set**	**Num.** Drivers	**Sources**	**Signals**	**Features**	**Best performing model**	**Result**
Meng et al. [113] 2006	Simulator	7	Virtual	Steering, Gas, Brake	FFT	HMM	80% for 7 drivers
Qian et al. [103] 2010	Simulator	7	Virtual	Gas, Brake, Steering	FFT, PCA ICA	SVM	85% for 7 drivers
Wakita et al. [99] 2006	Simulation, CIAIR	12, 30	Virtual Retrofitted	Gas/Brake Headway distance	Raw	Helly, Optimal Velocity GMM	Simulator: 81% 12 drivers Real car: 73% 30 drivers
Miyajima et al. [35], Nishiwaki et al. [36], 2007	Simulator, CIAIR	12, 276	Virtual CAN-bus Retrofitted	Gas/Brake Headway distance velocity	Raw, Cepstral	Optimal velocity GMM	Simulator: 89.6% Real car: 76.8% 276 drivers
Wahab et al. [100, 117] 2009	CIAIR	30	Retrofitted	Gas/Brake	GMM based, DWT based	Adaptive network-based fuzzy inference system (ANFIS)	95% for 10 drivers
Öztürk and Erzin [78] 2012	Uyanik	23	Retrofitted, CAN	Gas/Brake Headway distance	Cepstral	GMM	85.21% for 3 drivers
Del Campo et al. 2014 [37]	Uyanik	23	Retrofitted	Gas/brake pedal	Cepstral	MLP	84.6% - 3 drivers
Martinez et al. 2015 [38]	Uyanik	23	CAN, IMU, Laser-scanner	19 CAN, IMU based signals headway distance	time & freq. domain	ELM	90.7% - 3 drivers 76% - 11 drivers
Martínez et al. 2016 [39]	Uyanik	23	CAN, IMU, Laser-scanner	12 based on CAN, 6 based on IMU, headway distance	Cepstral Spectral etc.	ELM	96.95% for 3 drivers 84.36% for 11 drivers
Van Ly et al. 2013 [105]	Collected by UCSD	2	IMU	Gyroscope, accelerometer	Statistical	SVM	80% - 2 drivers
Zhang et al. 2014 [114]	Simulator	20	Virtual	Gas, Steering	Raw	HMM	85% for 20 drivers
Enev et al. [40] 2016	UCSD	15	CAN-bus	Powertrain, Car dynamics, Pedals, Steering	Statistical, FFT, etc.	RF	100% for 15 drivers
Hallac et al. [47] 2016	Audi AG & Audi Electronics	64	CAN IMU GPS	Steering, ERPM, X-Y acc., Gas, Brake, Throttle	Statistical, DWT,PCA	Random Forest	50.1% - 5 drivers 76.9% - two drivers
Kar et al. 2017 [41]	Independent	24	OBD, GPS	Open/close doors, Seatbelt, etc.	Timings and sequence	SVM	91% within 20s of entering vehicle
Phumphuang et al. [118] 2015	Independent		Smartphone	Lat./Lon. Acc. Speed, Heading, Lat./Lon., Alt	PCA	Eigen vectors	100% for 3 drivers
Zhang et al. [104] 2016	Independent	14	CAN-bus, Smartphone	GPS, Acc, Gas, ERPM	Statistical, Spectral, Cepstral	SVM	79.88 Accuracy - 14 drivers
Burton et al. 2016 [107]	Independent	10	Simulator	Velocity, Steering, Pedals		SVM	0.24 EER 10 drivers 0.147 EER 1 driver
Il Kwak et al. 2016 [42]	OCSLAB	10	CAN-bus	Vehicle operation Pedals & Steering	Statistical	Random Forest	0.995 Accuracy 10 drivers
Wang et al. 2017 [49]	Independent	30	CAN-data	VS, ERPM, GP, SWA, lat. lon.	Statistical	Random Forest	100%
Moreira-Matias and Farah [46] 2017	The first years study [120]	242 Families	Blackbox	GPS, IMU	Time-derived, POIs, aggressiveness	Decision tree (C4.5)	71.7% per family
Tanprasert et al. 2017 [108]	[119]	10	Blackbox, (GPS, IMU)	X-Y Accelerations.	-	SAX + MLP	81% for 10 drivers
Fung et al. 2017 [112]	CanDrive Study	14	GPS,CAN	Lat/Lon, VS	Acc/Decc events, Statistical	LDA	80% for 2/ 50% for 14 drivers
Wijnands et al. 2018 [48]	Insurance company	3000	Blackbox	Acceleration 1Hz, GPS	Event count	LSTM	Unclear
Jeong et al. [50] 2018	Own-data	4	CAN-Bus	ACC, Brake, SWA, SWR, VS, TPS, ERPM, Lat-Acc, Lon-Acc	- / Event Extraction	DL, CNN, LSTM	Up to 90% for 4 drivers
Ezzini et al. 2018 [43]	OCSLAB[42], HCI-Lab, UAH [121]	10 ,10, 6	CAN, EEG, Smartphone	GPS, Car state & dynamics,EEG	Raw	RF, ET	90%, 100%, 76%
Chowdhury et al. 2018 [110]	Independent	38	Smartphone	GPS, Jerk Acc, lat/lon Acc	Statistical Features	Random Forest	82.3 for groups of 4–5
Hernández Sánchez et al. [115] 2019	Independent	25	Smartphone	3-axis accelerometer	Recurrence Plots, Gramian Angular Summation Field, Gramian Angular Difference Field	DL, ResNet-50 + GRU layers	69.92% top-1
Lestyán et al. [109] 2019	Independent	33	CAN-bus	Gas, Brake, Velocity, RPM	Statistical	Random Forest	77% for 5 drivers
Virojboonkiate et al. [101] 2019	Independent	3–13	GPS, IMU	3-axis acceleration	Histogram	MLP, KNN, DT, RF, SVM	94–99%
Zhang et al. [44] 2019	OCSLAB[42]	10	OBD-II	All signals	Raw	DeepConv + GRU-Attention, LSTM-Attention, GRU, LSTM	0.9774 for 10 drivers
Chen et al. [45] 2019	OCSLAB[42]	10	OBD-II	10 out of 51 signals		NCAE + Softmax	99.65% 10 drivers

coefficients (MFCCs) are widely used in speech and speaker recognition [123]. The authors showed an identification rate of 76.8% using GMM for a field test with 276 drivers. This study obtains good results, but they were achieved using data from sensors that are not available on commercial vehicles. Pressure on the pedals is sampled at a rate of 1 kHz with high resolution, such data is not available on a standard vehicle; therefore, the same accuracy is not likely to be reproducible in a practical scenario.

There are several works that use the same dataset as ours (Uyanik [51]). But they are focused on a subset of 23 drivers that contain pedal pressure signals. Öztürk and Erzin [78], follow the same approach as Miyajima et al. [35]. However, the highest accuracy they achieve is 57.39% for 23 drivers, which is far lower than the reported accuracy by Miyajima et al. [35] (76.8%) for 276 drivers. We can explain this large gap in the accuracy by the low sampling rate of the signals in the Uyanik dataset.

Del Campo et al. [37] conduct a similar study as Öztürk and Erzin [78], but using a methodology based on MLP with a focus on real-time identification for which they also develop a hardware implementation. For a group of three drivers, they achieve an accuracy of 84%. In [38, 39] they propose to use extreme learning machine (ELM) (Huang et al. [102]) model and extract features from a broader set of signals from CAN-bus and IMU. Then a feature selection stage selects the more relevant features for identification. They obtain a 90% average accuracy for three drivers. Since the production cars are not equipped with pedal pressure sensors, the practicality of these works is questionable.

Meng et al. [113] propose to use dynamical models to represent driving behavior for driver identification purposes. They use three signals of SWA, gas, and brake pedals. They collect data using a simple driving simulator. FFT is used as a feature extractor, and they use the resulting feature vector to fit an HMM for each driver. They obtain 80% accuracy for 7 drivers. In another study using the same dataset, Qian et al. [103] compare FFT, independent component analysis (ICA) and principal component analysis (PCA) for preprocessing and feature extraction, and propose to use SVM for driver identifications. They identify that FFT is more suitable than the alternatives, and achieve an accuracy of about 85% for seven drivers. Zhang et al. [114] obtain steering and speed data from a simulator and model it using HMM. They obtain an accuracy of 85% for two drivers.

Burton et al. [107] propose to use one-class SVM for driver verification and a multi-class SVM solution for driver identification. They obtain an equal error rate (EER) of 0.24 for ten drivers and 0.147 for the driver verification task. Zhang et al. [104] use a window-based SVM for driver identification. They use data from OBD-II and smartphones collected from couples sharing a car and obtain 75.83% using only phone sensors, 85.83% using car sensors, and 86.67% with combined car and phone sensors.

Enev et al. [40] present a comprehensive work on driver fingerprinting. Although their focus is on the security and privacy implications of driver ID, they achieve good identification results. For 15 drivers they achieve 100% accuracy. taking advantage of three factors: *a*) a wide set of signals with high sampling rate (60Hz) from CAN-bus *b*) a comprehensive set of spectral and statistical features, and. They conclude that brake pedal and engine torque are among the most important signals for driver identification.

Van Ly et al. [105] take a different approach and use the IMU signals for driver ID. They manually segment the driving session into events such as acceleration, brake, turn, and then extract feature specifically for each event. They achieve an accuracy of about 80% for two drivers.

Some works look at specific situations. For example, Kar et al. [41], create a profile for each driver by modeling the order and the duration of actions they take after entering and before driving, actions such as opening and closing the door and fastening the seatbelt. They test this method on two groups of people and achieve good accuracy. While manufacturers have access to sensors needed to implement such an approach, this solution is highly vehicle-specific and challenging to deploy by third-parties. Moreover, it is not difficult to imitate someone else in this context. In another work, Hallac et al. [47] focus on the possibility of driver ID only using a single turn maneuver. They use simple statistical features in conjunction with discrete wavelet transform (DWT) as a feature extractor followed by PCA for dimensionality reduction, and use random forest (RF) as the classifier. Although they do not achieve high accuracy (50% for five and 76.9% for two drivers) they show that even in a single turn, there is enough information to discriminate between drivers (even with limited confidence).

There are a few works that focus on FCD. Moreira-Matias and Farah [46] use location and accelerometer data to identify family members. They use location data to construct clusters of locations and use the origin and destination clusters as a feature. They use other features such as excessive maneuvers extracted from the accelerometer, weekday, departure time, and trip duration. Then they use conventional ML algorithms to identify the driver and achieve accuracy of about 70%. Even though Wijnands et al. [48], focus on driver behavior change, they informally perform driver ID. They use a large dataset of 3000 drivers provided by an insurance company, that contains location and accelerometer data. From location data, they only use over-speeding events and the rest of the features are obtained by binning (thresholding) the accelerometer data. Chowdhury et al. [110] focus only on location data collected from smartphone, they estimate many secondary signals from speed and heading (e.g., jerk, lateral acceleration, and longitudinal acceleration). They compute 137 statistical features per trip and use a RF classifier to identify the drivers. For groups of 4 to 5 they obtain average accuracy of 82.3%.

In the past few years, we have witnessed an uptake in the number of research works on driver ID. Moreover, automotive, technology, and insurance companies are showing interest to this topic, either by providing datasets, funding or resources. Companies such as, Ford [49], Toyota [35], Samsung [42], Volkswagen and Audi [47], Yahoo, Microsoft, Technicolor [104], General Motors [41], Insurance Box Pty [48], Tata [110]. However, still a major hurdle in driver ID research is the lack of public datasets. There are only a few driving datasets which are limited. We could get access to Uyanik [51] dataset, but this dataset is not publicly available online. UAH-dataset [121] is one of the few publicly available datasets. However, it only contains data collected by smartphones from 10 drivers. The only dataset containing CAN-data is provided by OCSLAB [42] but has low sample-rate of 1 Hz. This particular dataset has already been used by the research community but with unexpected side effects. Most of the literature models driver ID as a classification problem. Some works such as [42, 44, 45] take every signal obtained from OBD-II and use them for classification with no regard for what they represent. As a result they mistakenly instead of modeling the driver behavior, they model the vehicle's state. Therefore we should be careful and make sure the signals we use are a function of the drivers and their behavior, not some environmental factor such as ambient temperature, humidity or slope of the road.

There are several shortcomings that are not yet addressed by the research community. The signals used in the current research are often not practical because they use sensors that are not available on production cars ([35, 37, 38, 78]). Some works require tight integration with the vehicle and are car-dependent [41]. Many use data from driving simulators or video games that may not adequately reflect real-world driving conditions [103, 107, 113, 114]. Computational complexity and ease of deployment is often ignored. More recent works often use ML methods such as SVM, bagging,or boosting methods with decision decision trees. Such methods are discriminative classifiers. This means that for each class (driver) we need to fit at least one model that discriminates the driver from the currently enrolled drivers. Such approaches are not scalable, because every-time a driver is enrolled, we need to retrain all the models.

5.6 What To Expect From The Following Chapters?

In the next four chapters, we aim to address some of the shortcomings in the state-of-the-art. First, we intend to introduce a driver ID solution that is practical, in the sense that it can be deployed to a wide variety of vehicles. We intend to achieve this by using the minimum number of signals that are widely available in cars on the market. Second, a driver ID solution should be scalable, enrollment of new drivers should require the least effort (no retraining of previous models). Third, the current methods often require lots of training data, and their performance drops significantly when the amount of data is low. Therefore, we intend to reduce the amount of required training data.

We take an incremental approach to develop driver ID methods that each will address some issues mentioned above. Chapter 6, proposes an ML method that only uses signals from CAN-bus, this means that such a method can be deployed on almost any car in the market, conditioned on having access to the car's internal network (CAN-bus). This method used five signals from the CAN-bus. Chapter 7 offers two improvements. We reduce the number of signals and the dimension of the feature vector per signal. We also address the scalability issue in the earlier approach. Chapter 8 tackles another issue, we further reduce the amount of data required for driver identification, this is only possible using deep learning techniques. We also introduce the concept of driver embedding under the name *deep driver* model, which we use for driver ID purposes. Chapter 9 looks into the case that no CAN-data is available and investigates the possibility of accurate driver ID using FCD, which in this case, is GPS data obtained from smartphones.

Genius is one percent inspiration, ninety nine percent perspiration.

Thomas A. Edison

6 Driver Identification Using Discriminative Classifiers

In this chapter, we propose a driver identification methodology based on discriminative models. In order to guarantee deployability, we exclusively focus on sensors that are already available in mass-market vehicles. Since we want this mechanism to provide driver identification in real-time, we run a classification algorithm in fixed intervals and to apply a decision function to define the driver's identity. The proposed mechanism extracts three category of features per each frame. The pattern matching algorithm is a discriminative classifier, which we can easily swap for any other classifier. We experiment with several classifiers and compare their performance. Lastly, we introduce two decision functions to make a final prediction over the frames in the ongoing session. In order to validate the proposed model, we use the Uyanik dataset presented in Chapter 2, which consists of 57 people driving a single car along a pre-defined route. We present the performance evaluation of the proposed mechanism in terms of the prediction accuracy. We also study the effect of the amount of training and prediction data on accuracy.

The remainder of the chapter is structured as follows. In Section 6.1, we present the proposed feature set, the classification algorithms, and decision-making rules to identify drivers. Then, in Section 6.3, we present evaluation results and corresponding discussions. Finally we summarize and present the next directions Section 6.5.

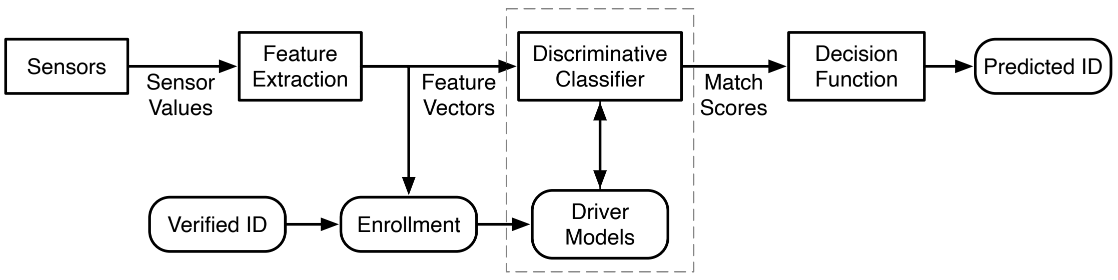


FIGURE 6.1: Block Diagram of the Driver ID method using discriminative classifier.

6.1 Methodology

The goal of driver identification is to associate a driving trace x to its corresponding driver. We limit the target drivers to a finite set and approach the driver identification problem as a supervised classification task. In this chapter, we only consider the closed-world scenario, which is the case that we have information on all drivers. For example, when a car is shared in a family and the goal is to identify which family member is the driver.

Therefore x is to be assigned to one of K drivers $\mathcal{C}_k$ where $k = 1, \ldots, K$. We search for a function h, which once trained is able to predict the corresponding driver ($\hat{y}$) for any unseen driving trace x. Since x consists of sequence of measurements $(x_1, x_2, \ldots, x_\mathcal{T})$, classical supervised ML methods are not directly applicable, therefore we apply the concept of *sliding windows* to make this possible [79]. In this approach the sequential data is broken down into smaller pieces (often called a window or frame) and then each piece is considered as a separate example. We use a *frame classifier* h_w to map each frame of length w into individual predictions. We use the term *frame* and reserve the term *window* to refer to a concept that will be introduced later in the chapter. Let $d = (w-1)/2$ be the half-width length of frame, then for a frame centered at time t, h_w makes prediction $\hat{y}_t$ based on the frame $\langle x_{t-d}, \ldots, x_t, \ldots, x_{t+d} \rangle$. Due to high sample-rate of driving signals it may not be feasible to make predictions for every frame, therefore, we make predictions for every $m = \lfloor w(1-r) \rfloor$ samples, where m denote *stride* and r denote the overlap ratio between two consecutive frames. This results in $N = \frac{\mathcal{T}}{m}$ examples.

In summary, each driving trace (x, y) is converted into N frames, then h_w is trained using feature vector $\mathbf{x}$ computed for each frame and original label y. Similarly to classify an unseen driving trace x, it is converted into N frames, for each frame feature vector $\mathbf{x}_i$ is computed and h_w makes prediction $\hat{y}_i$ based on $\mathbf{x}_i$. Finally $\hat{\mathbf{y}}$ results from concatenation of all $\hat{y}_i$. We require to assign x to only one driver therefore we define $\hat{y} = f(\hat{\mathbf{y}})$ as *decision function* which maps frame predictions to one single prediction for the whole driving trace.

Multiclass Classification

Since *driver identification* is performed among more than two drivers (generally a multiclass problem) therefore it constitutes a K-class problem. There are two general ways of doing multiclass classification, *one-versus-the-rest* (OvR), in which $K-1$ classifiers are used and each of them solves the task of discriminating between C_k and the rest of the classes. Another method is called *one-verus-one* (OvO), in which $K(K-1)/2$ binary classifiers are used to discriminate between each possible pair of classes. Each method has its own advantages, OvR is quicker to train, and more compact to store $K-1$ classifiers in compare to $K(K-1)/2$ in OvO, however OvO generally attains better accuracy (or perhaps any other performance metric of interest). OvO is also suitable in cases when the classifier is not well suited for larger problem instances, such as times there are too many examples, for example SVM is known to not scale well as number of examples increase.

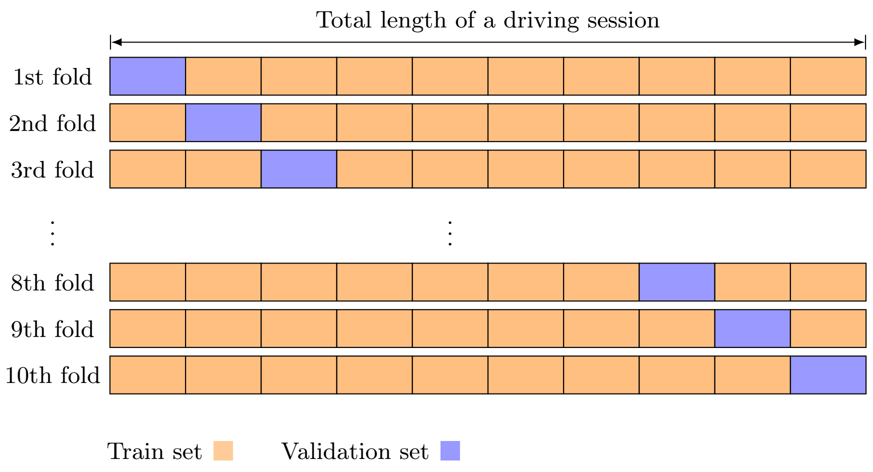


FIGURE 6.2: Cross-validation scheme.

Evaluation Method

In order to address applications with different requirements we perform evaluations for three driver set sizes of $K \in \{5, 15, 35\}$. Identification among drivers with similar driving styles is more challenging therefore we repeat for each K the evaluation $R = 30$ times and each time on a random subset of all drivers in dataset. For each instance, in order to utilize the entire driving traces $D = \{(x_i, y_i)\}_{i=1}^{K}$ for both training and testing, we employ a 10-fold cross-validation method. Each driving trace (x_i, y_i) is sliced into 10 segments of equal length and evaluations are performed under leave-one-segment-out train and test scheme. Therefore at each fold one segment from all driving traces is kept out of training process, then h is scored based on its prediction performance over the remaining segment, this scheme is depicted in Figure 6.2. Note that we always train on same segments for all drivers and test on the same segment, for example we train on

TABLE 6.1: Selected Signals for Driver Identification

Sensor	Derivatives 1st	Derivatives 2nd	Cepstral & Spectral
Percentage Gas Pedal (PGP)	Yes	Yes	Yes
Steering Wheel Angle (SWA)	Yes	Yes	Yes
Vehicle Speed (VS)	Yes	Yes	No
Engine RPM (ERPM)	Yes	Yes	No
Yaw Rate (YR)	Yes	Yes	No

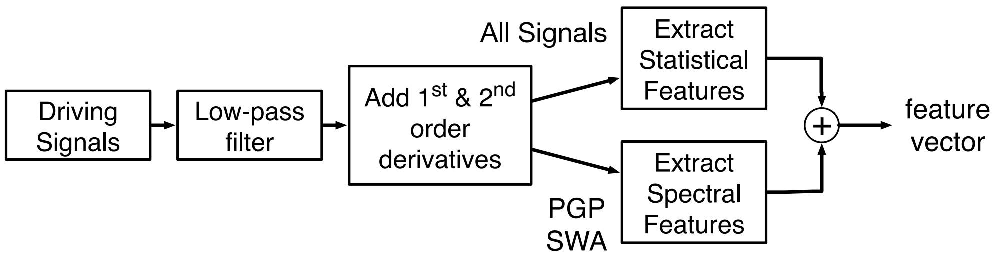

Driving Signals
Low-pass filter
Add 1st & 2nd order derivatives
All Signals
Extract Statistical Features
PGP SWA
Extract Spectral Features
feature vector

FIGURE 6.3: Feature extraction block diagram.

segments 1–9 and test on the 10th segment. Since all drivers drive through the same route, even though some drive slower than the others, we assumed in this case training and testing would be done on similar parts of the trip for all drivers, therefore resulting in a fair comparison.

We use accuracy to score h at each fold, which is defined as below:

$$\alpha' = \frac{1}{K}\sum_{i=1}^{K} \mathbb{1}(y_i = \hat{y}_i) \tag{6.1}$$

where $\mathbb{1}$ is the indicator function, $\hat{y}_i$ and y_i are respectively prediction and true driver for i^{th} driving trace.

The overall accuracy score for h is then obtained by calculating the mean accuracy of all cases as follows:

$$h_{score} = \frac{1}{R \cdot L}\sum_{j=1}^{R}\sum_{i=1}^{L} \alpha'_{i,j} \tag{6.2}$$

where $L = 10$ is number of cross-validation folds and $R = 30$ is number of repetitions. In the rest of the chapter we refer to this score as accuracy.

6.1.1 Preprocessing and Feature Extraction

In this work we only use 5 signals, all available in the car's internal network (CAN-bus). This decision was made for two reasons, to use the most trivial signals that are available in majority of cars on the market, and to reduce the complexity of the system which will affect the deployability. A signal refers to a stream of sensor measurement values. Two kinds of signals are used for feature extraction: *a*) original signals *b*) derived signals. The original signals are the sensor values from CAN-bus, while derived signals are the first and second derivative of the original signals. To motivate this choice let us take vehicle speed as an example: its first and second derivative represent longitudinal acceleration and jerk, which have shown to be representative of particular driving styles [124], therefore we hypothesize that these signals are useful for discriminating drivers. Figure 6.3 shows the various stages that the original signals go through before the feature extraction step. The 5 original signals used in this study are listed in Table 6.1. First a low-pass filter is applied to original signals to remove high frequency noise and smoothen the signals. Then for each signal we use the Savitzky-Golay [82] filter to derive their first and second order derivatives which results in two additional signals per each original signal. Then, all the signals go through a framing function which takes two parameters, w to determine the frame-size and r to specify amount of overlap between two consecutive frames. At this point each signal is broken down into frames of size w and is be supplied into the feature extraction stage. Next stage is where the features are extracted from each frame and are used for training and later on for prediction in the identification process.

Three kinds of features are extracted from each frame:

Statistical features a set of descriptive statistics is selected to be most representative of the distribution of sensor values covered by the frame. This set includes minimum, maximum, mean, median, standard deviation, kurtosis, skewness. This category of features is extracted from all the signals.

Cepstral and spectral features Cepstral features have various applications and are most widely used in speech and speaker recognition [125]. Studies have shown that cepstral features are suitable for driver identification [35, 37, 78]. Use of spectral features for classification was first proposed by Agrawal et al. [126] in the context of querying databases. The idea of using cepstral feature for identification comes from speech modeling. In speech modeling we assume that the vocal cord excitations are filtered with the vocal tract. Vocal cord vibrations are represented by the fine structures of spectrum and the vocal tract by the spectral envelope. Unique variations in the vocal tract, result in distinct spectral envelope. The analogy to driving is that action of depressing the pedal or steering is filtered by driver model—since each driver has unique anatomical characteristics—we can assume that different drivers express distinct spectral envelopes which we can use to drivers by modeling the cepstral coefficients. By keeping the first $k = 8$ cepstral coefficients and setting the remaining coefficients to zero we smooth the spectrum and isolate the spectral envelope (see [35] for more details). We extract spectral features only from two signals, PGP and the SWA and keep the first $k = 8$ cepstral coefficients as features. As it is shown in Figure 6.5 signal from each window frame is pre-emphasized[1] to enhance higher frequency components, we compute log-magnitude-spectrum of the signal. From here we perform two different set of operations to obtain

[1] $y[n] = x[n] - \beta x[n-1]$

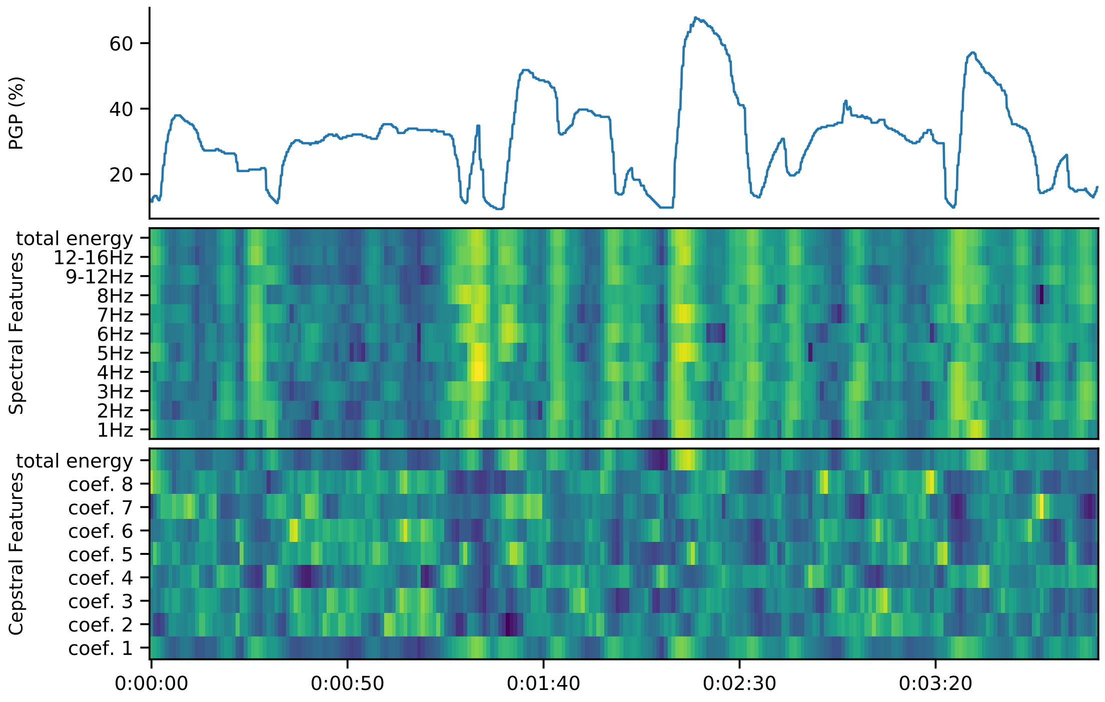


FIGURE 6.4: PGP signal and visualization of corresponding features. Each row of spectral and cepstral features are independently normalized to highlight the variations in features. Bright colors have larger values. Swings in PGP results in higher values in spectral features.

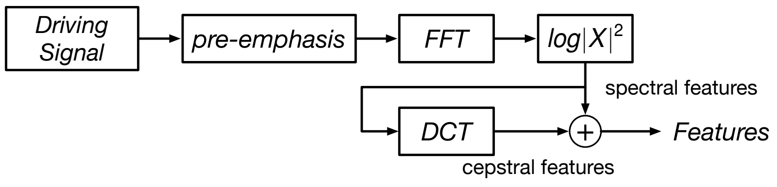


FIGURE 6.5: Block diagram of computation of spectral and cepstral features.

two groups of features, spectral features and cepstral features. First we multiply the frame with a Hamming window[2][127] of length w to limit the signal and also reduce the edge effects. Spectral features are derived from the spectra which is obtained from the following equation.

$$X = \log(|\mathcal{F}\{xw\}|^2) \tag{6.3}$$

where $\mathcal{F}$ is the Fourier transform and w is the Hamming window. log-power-magnitude of the spectra for each 1Hz range up to 8Hz however, since higher frequencies have smaller magnitude we compress them in larger ranges, in this case 8–12Hz and, 12Hz and above. We also include the sum of all energies as a separate feature, therefore, which sums up to 11 spectral features. To obtain cepstral features we take the discrete Cosine transform (DCT)[128] to log-power-magnitude of spectra. We define $c(k)$ to be the first k cepstral

[2]Hamming window is defined as: $w(n) = 0.54 - 0.46\cos(\frac{2\pi n}{M-1}) \quad 0 \leq n \leq M-1$

coefficients and higher order coefficients are discarded.

$$c(k) = \mathcal{DCT}\left\{\log(|\mathcal{F}\{xw\}|^2)\right\} \tag{6.4}$$

where $\mathcal{DCT}$ is discrete Cosine transform. In this work, we only keep the first $k = 8$ cepstral coefficients as features. Like spectral features, we also consider the sum of energy among the cepstral coefficients as a feature. This will leave us with 9 cepstral features. This results in a feature vector of dimension 20 per signal.

6.1.2 Classification Algorithms

We select four classification algorithms to be used as *frame classifier* h_w. We benchmark them and pick the best performing algorithm to conduct further experiments. In our selection we favored ensemble methods because it has been shown that they generally achieve better predictive performance [129]. The following is the list of classification algorithms used in this study along with their corresponding hyperparameters.

AdaBoost (AB) - AdaBoost as introduced by Freund and Schapire [85] suggests use of collection of weak learners, as the number of weak learners increases the algorithm increases the weight for examples that are more difficult (are currently misclassified) therefore improving the performance. In the implementations we use the scikit-learn library [86] which uses the AdaBoost-SAMME algorithm [87]. We use a decision tree with maximum depth of 1 as weak learner [88]. 1000 decision trees as weak learners, learning rate = 0.75.

Gradient Boosting (GB) Gradient boosting, iteratively combines weak learners into a single strong learner. At each iteration GB fits a tree to correct the mistakes of the model of previous iteration (and all the iteration before). Therefore in principle Gradient boosting fits models to the residuals of the previous iteration. These residuals equivalent to negative of the gradients [89]. 1000 estimators, maximum depth = 15, min samples split = 5, max features = 20.

Random Forest (RF) Random Forests as introduced by Breiman [90]. As the name suggests it consists of a combination of tree predictors such that each tree is train on a subsample of the examples, this is also known as bagging. Moreover to further reduce the correlation between the trees and avoid overfitting, each tree is fitted on a subset of features. We used the implementation from scikit-learn package [86]. With parameters 1000 estimators, maximum depth = 30, max features = 20, minimum samples split = 3, bootstrap = True, class weight = balanced, criterion = entropy

Support Vector Machine (SVM) Lastly we step away from tree based algorithms and attend to once upon a time king of classifiers, SVM. SVMs for some time were known to be the best classifiers available. Linear support-vector machine was introduced by Vladimir Vapnik in 1965. SVMs are binary classifiers, in the Linear case SVM finds the a maximum-margin hyperplane that separates the two class. Cortes and Vapnik [91] in 1995 proposed to use the kernel trick [92] and extended SVM to solve non-linear classification problems. In this approach instead of dot products in SVM a non-linear kernel function is used. Therefore algorithm is able to fit maximum-margin hyperplane in a transformed space, which allows for separation of non-linear feature boundaries. In the implementations we used the SVC classifier from scikit-lean library [86] which uses LIBSVM library [93]. RBF kernel, γ=0.001, C = 1000, class weight = balanced.

For ensemble learners in order to have a balance between prediction performance and execution time we choose 1000 as the number of weak learners/trees. In our experiments we use the implementations from the scikit-learn software package, all other parameters are set to their default values as of scikit-learn version 0.18.1 [86].

6.1.3 Decision Functions

The decision function f determines the final prediction $\hat{y}$ based on window predictions $\hat{\mathbf{y}}$. We define for each window k a vector $\hat{\mathbf{y}}_k = (\alpha_1 \ \cdots \ \alpha_c)$ where $\alpha_{k,j}$ corresponds to the classification score on the window k for driver j. The term *classification score* refers to the output of the classifier, which depending on the classifier could refer to probability or distance to hyperplane (in case of SVMs). Below we introduce two decision functions to obtain $\hat{y}$ and evaluate their performance later in the chapter:

1) Majority vote (MV) For each temporal window k, we find the identifier of the driver with largest classification score $I_k = \arg\max \alpha_i$. Then We assign to each temporal window a ranking score vector $\mathbf{RS_k} = \left(\beta_1^k, \ldots, \beta_c^k\right)$ where $\beta_{I_k}^k = 1$ and all other $\beta_i^k = 0$. Finally $\hat{y}$ is the identifier of the driver who most frequently obtains the highest classification score, i.e., $\hat{y} = \arg\max_i \sum \beta_i^k$

2) Maximum score (MS) We set $\hat{y}$ to be the identifier of the driver that has the largest cumulative classification score, i.e., $\hat{y} = \arg\max_i \sum_{k=1}^{N} \alpha_i$

To motivate the introduction of the two decision functions let us take the following example which consists of classification scores of three drivers for five windows:

$$(\hat{y}_k)_{k=1}^5 = \begin{pmatrix} 0.15 & 0.40 & \mathbf{0.45} \\ 0.18 & 0.39 & \mathbf{0.43} \\ 0.27 & 0.34 & \mathbf{0.39} \\ 0.06 & \mathbf{0.73} & 0.21 \\ 0.09 & \mathbf{0.81} & 0.1 \end{pmatrix}$$

In MV we simply count how many times each driver has the highest score and choose the most frequent as prediction:

$$\hat{y} = f_{\text{MV}}((\hat{y}_k)_{k=1}^5) = \arg\max_i \left(0 \ 2 \ \mathbf{3}\right) = \mathcal{C}_3$$

MV is sensitive to miss-classifications at h_w. Suppose d_2 is the true driver, for the first three windows $\mathcal{C}_2$ and $\mathcal{C}_3$ are scored very closely but at window level choice is always with $\mathcal{C}_3$. Looking at the next two windows and the whole picture $\mathcal{C}_2$ is more likely to be the correct prediction. Next we compute the prediction based on MS:

$$\hat{y} = f_{\mathrm{MS}}((\hat{y}_k)_{k=1}^{5}) = \arg\max_i \, (0.75 \; \mathbf{2.67} \; 1.57) = \mathcal{C}_2$$

While MV falls short at this example MS handles such instances more robustly.

6.2 Feature Analysis

We perform some feature analysis to obtain some insights into effectiveness of the proposed set of features. We mainly use tree-based feature importance as a measure for feature importance [130]. Particularly the variant implemented in scikit-learn library, this approach often called *mean decrease impurity*, and defined as total decrease in weighted node impurity averaged over all trees in the ensemble. Total decrease in node impurity is weighted by the probability of reaching the node, which is approximated by the proportion of examples reaching that node. Since the features are extracted on a per-frame basis, there is an interaction between features and the frame-size, therefore we perform our analysis for various frame sizes. Since most features represent some aspects of the vehicle dynamics there are features that positively or negatively correlate with each other. Our goal here is to study the utility of each feature for driver ID. To obtain the most efficient feature representation dimensionality reductions techniques such as PCA can be used to eliminate collinear features.

6.2.1 Statistical Features

In order to evaluate effectiveness of statistical features we run a set of experiments using only the statistical features of all original and derived signals and therefore exclude spectral and cepstral features. The results are depicted in Figure 6.6. We can see min, max, range, standard deviation perform well in compare to other features and their effectiveness is stable across various frame-sizes. On the other hand we can see features like mean and median, lose their importance as the frame-size increases. This could be caused by the fact that the longer the frame, the closer the mean/median of a feature is to global mean of that signal value which is probably the same for every driver. For example SWA should have an average of 0 for every driver, as the mileage increases. This interpretation however is not valid for every signal, for instance, mean of ERPM should be quite informative as it reflects driver's preference to drive on a specific range of engine revs—some prefer lower engine revs and switch gears earlier than other drivers that prefer to driver on higher revs and switch gears later (in case of increase in speed).

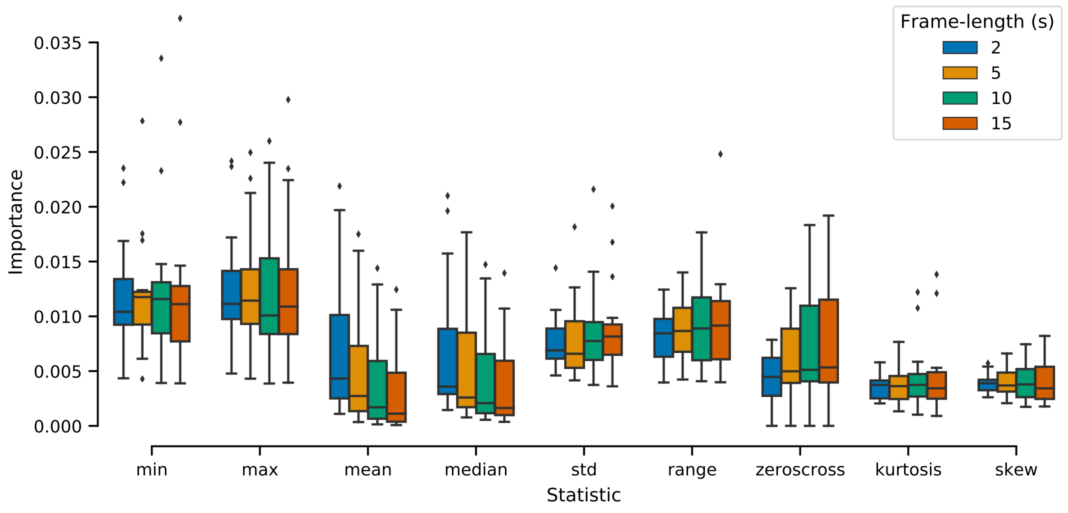


FIGURE 6.6: Feature importances for statistical features. For this experiments spectral and cepstral features are excluded and only the statistical features from all original and derived signals are used. Some features such as min, max, and range maintain their relative high importance for all frame-sizes. Some features such as mean and median lose their effectiveness for larger frame-sizes. It is also interesting to see # of zero crossings, becomes more importance for larger frame-sizes, as it is less likely to observe significant number of zero crossing in for example 2s of driving data.

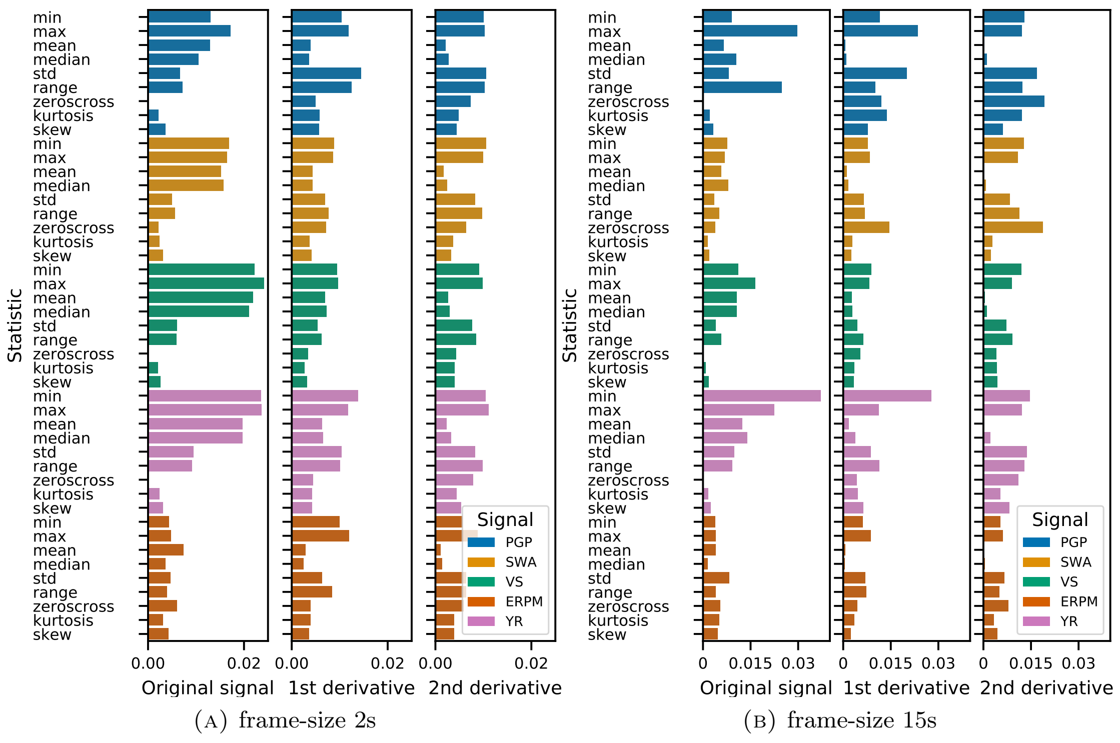


FIGURE 6.7: Statistical feature importances for original and derived signals, for frame-sizes of 2 (left) and 15 seconds (right). We can see that there is large variation between the two frame-sizes. For frame-size of 2 seconds, mean max and median of YR, VS, SWA and PGP are among the top features. For frame-size of 15, min, max of YR, range and max of PGP, max and std. of PGP', min of YR' and num. zero crossings of SWA" and PGP" are among the top features.

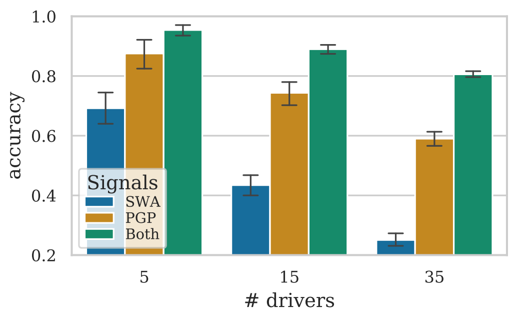


FIGURE 6.8: Accuracy with respect to feature-set. This experiment shows accuracy in three cases, PGP or SWA alone or Both toghether. PGP alone shows good results but SWA by itself is not a good discriminator, however together they show improved performance. This improvement also shows itself as reduction in variance of accuracy, which means the system performs more robustly using two signals as opposed to PGP only.

Figure 6.7 shows how the effectiveness of statistical features changes with the frame-size. For small frame-length simple features like min, max and mean of PGP, SWA, VS and YR consistently have high feature-importance. But when the window size is larger, the good features are more spread out. For PGP, max and range are the best features, as well as standard deviation and max of its first derivative. Min and max of YR and its first derivation are good features as well. ERPM is no longer a good feature. Number of zero crossings of SWA is a good feature, this is in contrast to smaller frame-lengths, as in small frame-sizes there is either no or one zero crosssing. VS is also losing its importance, except its maximum. But in general PGP and YR seem to be the best features. We can conclude that depending on the frame-size, the informativeness of features varies.

6.2.2 Cepstral and Spectral Features

Some works in the literature such as [35, 37, 78] show that the cepstral features of gas and brake pedal pressure signals improve the driver ID accuracy. Such signals are obtained by fitting external sensors under the pedals and are not available in production vehicles. Instead, we choose PGP and SWA that are available on unaltered vehicles from the car's internal computer network. We choose SWA because, similar to the pedals, the steering wheel is also operated directly by driver and its movements potentially reflects some unique characteristics of driver.

In order to validate the positive impact of the SWA signal on driver identification, we conduct three experiments. First, we limit the features only to cepstral and spectral features of PGP, then only to cepstral and spectral features of SWA, and lastly cepstral and spectral features of both PGP and SWA. Results are depicted in Figure 6.8, as one can see when each signal is used alone, PGP yields better results than SWA, however when used together results significantly improve (8.98 to 36.56%) from the best single signal case, this is particularly noticeable when the number of drivers increases. This is probably because some of the drivers operate the gas pedal or the steering wheel similar to one another; the use of PGP and SWA at the same time helps the classifier to discriminate the drivers in this ambiguous cases.

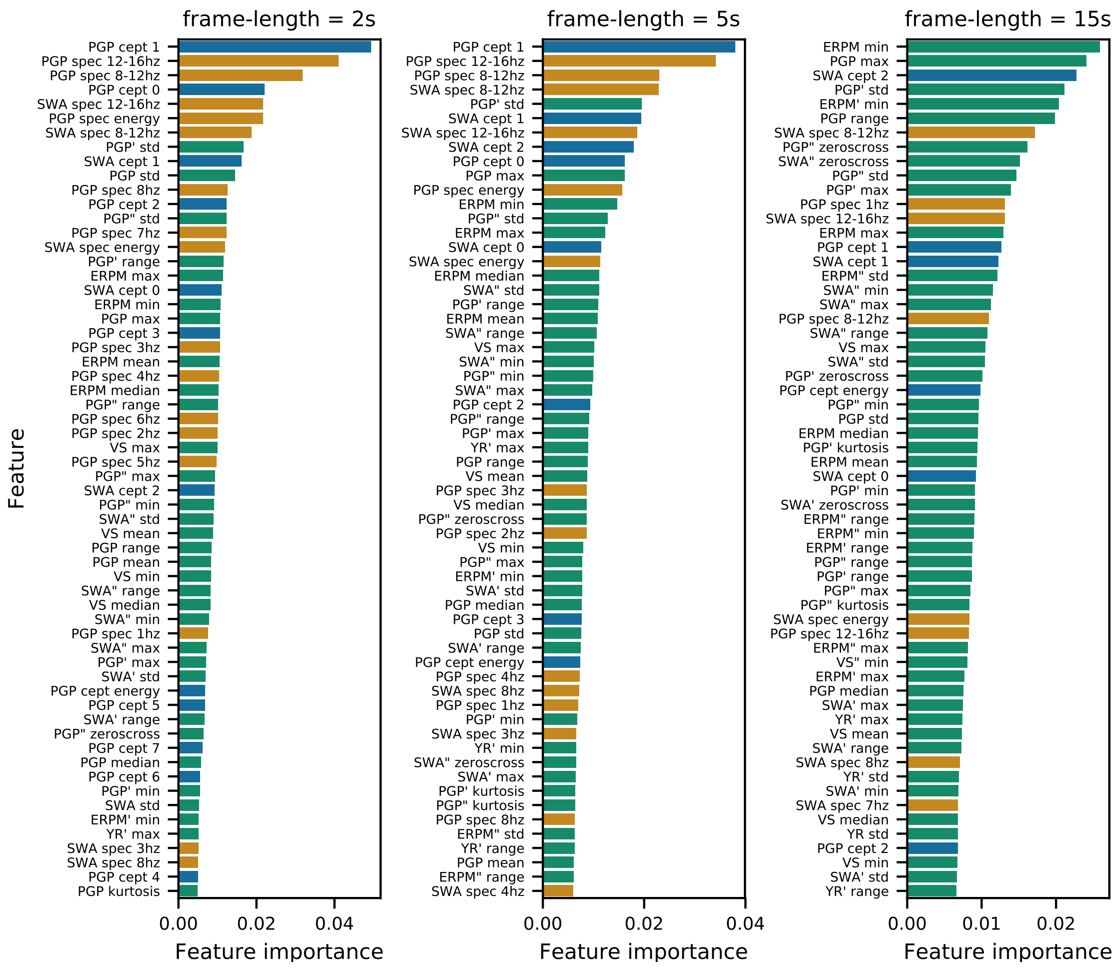


FIGURE 6.9: Feature importance for 3 different frame-lengths of 2, 5, and 15 seconds. The colors indicate the category of features, begin spectral, cepstral or statistical. An interesting observation is that for frame-size=2s there are quite a few spectral or cepstral feature among the most important features, but as the frame-length increase they give their position to statistical features.

6.2.3 Overall Feature Importance

In another experiment we evaluate the feature importance for all the features, which includes the statistical of all signals, and spectral and cepstral of SWA and PGP. The results for frame-sizes of 2, 5, and 15 seconds are depicted in Figure 6.9 (only top 60 features are included in the plot). The colors indicate the feature category, green corresponds to statistical features, blue for cepstral and the golden color is for spectral features. An interesting observation that can be made from this figure is how spectral and cepstral features dominate the top features for small frame-size of 2, and to some extent for frame-size of 5 seconds, but on the contrary when the frame-size is 15 seconds, statistical features occupy majority of the top features.

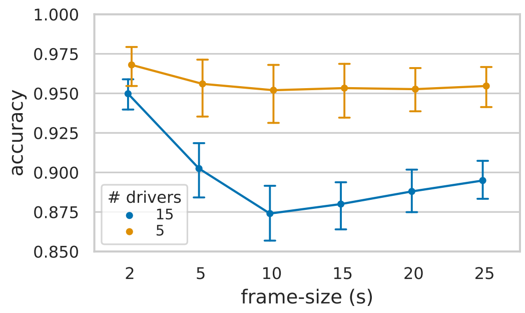


FIGURE 6.10: Accuracy with respect to frame-size. Smaller frame-sizes favor spectral features, meanwhile statistical features are more informative when they are obtained for larger frames. As the frame-size increases, spectral features lose their utility but statistical feature become more in formative and improve the accuracy—this effect is more pronounced for 15 drivers.

6.2.4 Frame-size Analysis

Since the feature vectors are extracted from frames, a key parameter in our method is the choice of w and r (stride). We perform experiments for frame sizes of [1] from 2 to 25 seconds, with r fixed at 1s. According to results shown in Fig. 6.10, as w increases accuracy decreases until reaching its minimum at 10 seconds then starts to slowly improve. We hypothesize that this is a symptom of interactions between the features and frame-size, as we showed earlier, statistical features perform better for larger frame-sizes while spectral and cepstral features perform better for small frame-sizes. Therefore it is possible that at 10s all feature categories are at their worst discriminative power, and moving away from that spot, with smaller frames, spectral and cepstral features compensate, and for larger frames, statistical features dominate.

We also investigate the impact of stride (r). Experimenting with stride of 0.25 and 1 seconds, we observed that smaller stride generally improved the performance in some cases by 1-2 percentage points, but at the same time it will increase the number of examples therefore is computationally expensive. Since we intend to maintain reasonable training times and considering that the gain in accuracy is not significant we will continue using the stride of 1s. Earlier works such as [35] and [78] use window sizes of 0.8s and 0.32s, but they do not report how they arrive at these numbers. Enev et al. [40] perform a similar experiment but they conclude that 3s results in the best performance, which is close to our findings. This discrepancy however should not be seen as an issue, since there are numerous factors involved in obtaining the optimal frame-size, including, the choice of signals, features, sample-rate, and even the drivers and the driving route.

The choice of frame-size and stride is highly application dependent. Note that in this experiment we used a fixed stride, in cases that computational resources are limited, one could employ larger frame-sizes in conjunction with larger strides, and therefore significantly reduce the number of examples, hence reduce the amount of compute required. Such approach would most likely negatively affect the performance but for some use-cases such compromise may be acceptable. For example in case a fleet manager wants to validate the actual driver for a long trip a fairly large frame-size and small overlap

[1] Frame sizes of length 2, 5, 10, 15, 20, 25 Seconds

would be sufficient. On the other hand if it is desired to identify the driver as quickly as possible, for example to customize the driving experience, then having a small stride and reasonably small frame-size is preferable. One could even go one step further and choose the stride adaptively.

6.3 Experiments

In this section, we present results of the experiments and their corresponding discussions. Each subsection is focused on a certain component of the proposed method, therefore in order to avoid repetition unless otherwise is stated experiments are performed using configurations introduced in Section 6.1, with Adaptive boosting trees (AB) OvO as h_w, MS as f, $w = 2s$ and $r = 1s$.

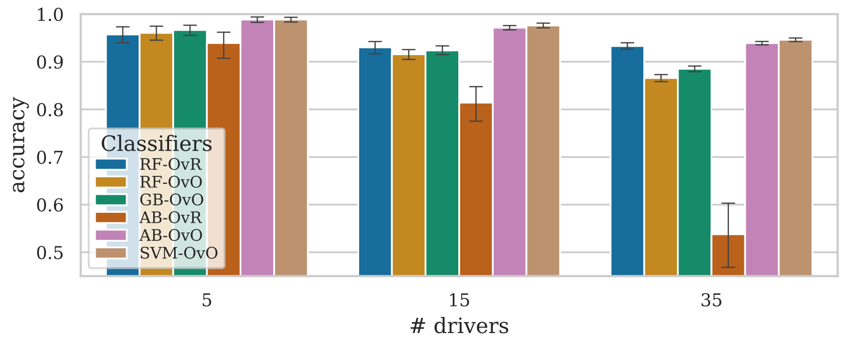


FIGURE 6.11: Classifier identification performance comparison. The SVM and AB used in one-versus-one configuration provide the best results.

6.3.1 Classifier Benchmarks

Identification results for various classification algorithms are depicted in Figure 6.11. The OvO approach generally provides better performance than OvR. The best performance belongs to SVM-OvO, with 98.86%, 97.62%, 94.61% accuracy for groups of 5, 15, and 35 drivers, respectively. The second best performance obtained with AB-OvO, with 98.86%, 97.15%, 93.92% accuracy for groups of 5, 15, and 35 drivers, respectively.

We suspect that with further hyperparameter tuning similar results, can be obtained with other classifiers such as gradient boosting (GB), and RF. In the end the right choice of classifier depends on other aspects such as available computation, training time, and dependability on hyperparameter tuning. We choose to continue the rest of our experiments with AB because although it produces comparable results, it is faster than SVM. The the results achieved using AB are better than earlier works in the literature, except for [40] in which authors achieve 100% accuracy for 15 drivers. However, they use 15 signals from CAN-bus and perform spectral analysis on all signals, while we only use 5 signals and extract spectral and cepstral features from two of them. It is worth mentioning that in this work we keep the study realistic therefore we do not selectively filter out the sections of trip that are not the most informative. For example in [37] authors only use the reference-driving segment of the trips, or in [35] they only consider the times that the car is on the move. Such assumptions although suitable for preliminary studies they do not reflect the real-world conditions.

Table 6.2: Comparison of decision functions

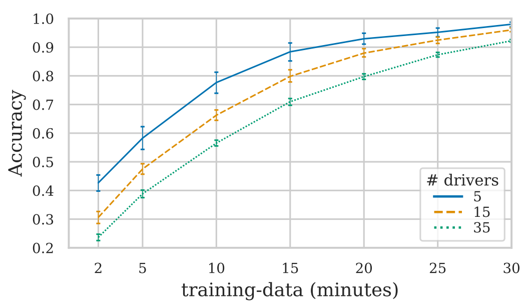
# drivers	Test-size	Decision function accuracy			
		MV		MS	
	Minutes	mean	std	mean	std
5	1	0.961	(0.036)	0.960	(0.031)
	2	0.972	(0.030)	0.969	(0.026)
	3	0.980	(0.018)	0.981	(0.019)
15	1	0.908	(0.027)	0.908	(0.025)
	2	0.933	(0.021)	0.934	(0.023)
	3	0.961	(0.018)	0.957	(0.018)
35	1	0.880	(0.019)	0.866	(0.018)
	2	0.918	(0.017)	0.910	(0.016)
	3	0.936	(0.013)	0.922	(0.014)

6.3.2 Decision Functions

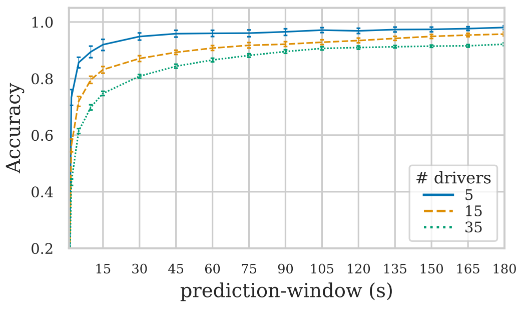
The final stage of the proposed methodology is the decision function f. To investigate performance of each decision function we perform several experiments using MV and MS and compare the results, for these experiments we use AB-OvO, as it was shown to yield the best identification performance. Results are summarized in Table 6.2. We can see that MV generally provides better performance, but not a significant improvement with respect to MS. We conducted a similar experiment for other classifiers, generally MS works better when classifiers are trained in OvR scheme, and MV provides better performance in case OvO scheme is used. The implementation of MS assumes calibrated probability outputs [131] from the classifier h_w which may not be provided by the classifier or require additional computational costs to be obtained. On the contrary, MV only requires the prediction labels of each frame.

6.3.3 Sensitivity Analysis of Training-data

The amount of training data required for training ML models is quite important. Especially in driver identification we wish to be able to train our models with as little data as possible. Therefore we conduct experiments in which we vary the amount of training data used by the model. For example for each fold, if the total available data is say, 30 minutes, we only select the first 5 minutes for training the model. For this experiment we use 3 minutes of driving data for prediction. In these experiments we test 6 instances $\{5, 10, 15, 20, 25, 30\}$ minutes. The corresponding results are depicted in Figure 6.12a. We can see accuracy improves with increase in amount of training data. For 5 drivers



(A) Variable train-data



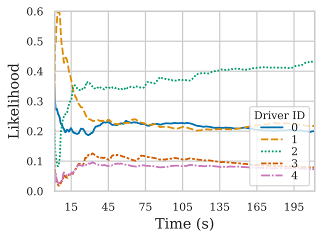
(B) Variable prediction-data

FIGURE 6.12: On the left we can see how amount of training-data affects the performance. 20 minutes of training data is required to achieve above 80% accuracy and 25 minutes to achieve 90% accuracy.

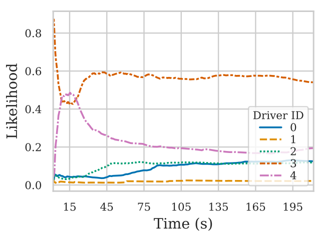
15 minutes of driving data is needed to cross 90% accuracy, while to achieve the same accuracy for 15 drivers about 20 minutes and for 35 drivers about 25 minutes of driving data is needed.

6.3.4 Sensitivity Analysis of Decision Window

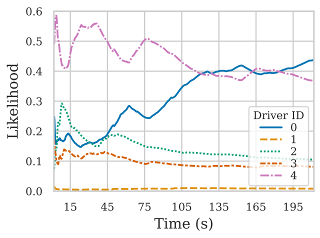
Another important factor in driver identification is the amount of data needed to make an accurate prediction. In an experiment we keep the amount of training data fixed (in this case all available training data) and vary amount of data used to make a prediction. We use range from 0 to a 180 seconds. The results are shown in Figure 6.12b, accuracy in all cases quickly reach 80% with 30 seconds of data. As it is expected for smaller number of drivers prediction accuracy improves very quickly and plateaus with about 1min of driving data. On the other hand for 15 or 35 drivers it takes longer to get to accuracy of over 90% yet even with 2 minutes of data even though improvement slows down it appears that results would still improve if more data was available.



(A) Target driver 2



(B) Target driver 3



(C) Target driver 4

FIGURE 6.13: Prediction over time. Visualization of normalized cumulative prediction score for three cases. The driver with largest likelihood is predicted. *(A)* is case where the system corrects itself after about 30 seconds. *(B)* shows an example of incorrect prediction. *(C)* shows a case that although in the beginning the prediction is correct, after larger amounts of data, we result flips to incorrect prediction.

6.3.5 Real-time Identification

We also investigate how the incremental addition of more data samples impacts the prediction performance. Figure 6.13 shows three identification cases. In these plots the vertical axis denotes normalized cumulative prediction score for each driver and horizontal axis denote time in seconds. Better prediction score should be interpreted as higher certainty for the corresponding driver therefore as we advance in time and more examples are available for prediction, we expect higher certainty for predicted driver. Case A shows that although for the first $10\,s$, expected prediction could be wrong ($\mathcal{C}_1$ instead of $\mathcal{C}_2$) as more samples are collected, the prediction changes in favor of the correct driver. Case B shows an example in which the very first 5 seconds predict the correct driver, then at around 15 seconds, system makes a mistake, then again as more examples are obtain reverts back to the correct prediction $\mathcal{C}_3$. Case C shows an unexpected outcome, the model makes correct predictions right from the beginning and with high certainty, but after about 2 minutes, prediction becomes uncertain and finally make the wrong prediction ($\mathcal{C}_0$ instead of $\mathcal{C}_4$).

6.4 Discussion

We presented an ML-based approach to driver ID. We chose a sliding window approach because it is flexible. One can set the prediction interval according to the available processing power. In this approach, there is no need to maintain a long history, and only the past prediction scores are enough to make a decision for the latest frame. Moreover, we can start making predictions with as little as one frame, and update the prediction as more data is collected.

We did not make any assumptions about the data, we used all data as it is, without removing any of the data. Some works use data from specific parts of a trip, such as turns, or remove parts of the trip that the car is not moving. We made this decision to show that this method does not require extensive pre-processing and therefore suitable to be used in practice. It easy to remove the pieces of information where the car is not moving. However, it is not feasible to correctly detect and align each turn in production, something that is done in [47].

We observed that statistical features are more effective when the frame-size increases—over 10 seconds but less than 45—a better way of constructing feature vectors would be to use two frame-sizes, a shorter frame to compute the spectral and cepstral features and a longer frame for extracting the statistical features. Another idea would be to compute spectral and cepstral features over small frames, and instead of their values, use their statistics in larger frames (along with other statistical features). These ideas can be explored in future work.

TABLE 6.3: Summary of the results

train-data	test-data	# Drivers - Accuracy mean (std)					
Minutes	Minutes	5		15		35	
5	1	0.575	(0.106)	0.438	(0.061)	0.367	(0.032)
	2	0.589	(0.109)	0.470	(0.062)	0.415	(0.035)
	3	0.594	(0.107)	0.494	(0.061)	0.431	(0.035)
	4	0.597	(0.116)	0.509	(0.063)	0.439	(0.038)
10	1	0.743	(0.091)	0.617	(0.044)	0.537	(0.030)
	2	0.775	(0.100)	0.674	(0.053)	0.600	(0.028)
	3	0.788	(0.096)	0.697	(0.061)	0.624	(0.025)
	4	0.795	(0.101)	0.714	(0.051)	0.640	(0.024)
20	1	0.905	(0.058)	0.796	(0.047)	0.721	(0.021)
	2	0.921	(0.055)	0.857	(0.038)	0.790	(0.023)
	3	0.929	(0.054)	0.888	(0.032)	0.825	(0.021)
	4	0.943	(0.048)	0.908	(0.031)	0.851	(0.020)
30	1	0.953	(0.040)	0.886	(0.027)	0.839	(0.018)
	2	0.963	(0.032)	0.924	(0.025)	0.888	(0.016)
	3	0.977	(0.024)	0.955	(0.018)	0.917	(0.012)
	4	0.983	(0.018)	0.970	(0.015)	0.942	(0.010)

A significant limitation of the proposed methods is due to use of discriminative models. Discriminative classifiers learn to discriminate drivers; a driver's model or template is obtained by training on driving examples from every driver enrolled in the system. Therefore, if we deploy a driver ID solution for K drivers, and decide to enroll a new driver to the system, we would need to fit at least $K + 1$ new models and update the models deployed on all vehicles. This is costly both in terms of computation and data transmission (incurred by delivering $K - 1$ new models to each car).

6.5 Summary

In this chapter, we proposed a mechanism for driver identification based on driving signals currently available in cars on the market. Therefore there is no need for any external sensors. In particular, we use simple statistical features from the following 5 signals and their first and second order derivatives: PGP, SWA, VS, ERPM, YR. Additionally we also take advantage of more complex cepstral and spectral features that are only obtained from PGP and SWA. The proposed methodology, uses an iterative sliding window approach, and extract features from each frame and a classifier makes predictions per frame. The the final driver identification result for an on-going driving session is provided through a decision function that take into account the classification scores of previous iterations.

We proposed an evaluation study based on a large driving dataset. We performed extensive feature analysis to evaluate importance of each feature and signals and studied how frame-size affects the features and prediction performance. The results show that adding the spectral and cepstral features of SWA increases the identification performance by 36.56% in average—for 35 drivers—in compare to only using the same features from PGP. We experimented with a variety of classifiers shown that Boosting methods and SVMs are suitable for this problem. Some of the best results have been achieved using

AB-OvO with $98.86\%, 97.15\%, 93.92\%$ accuracy for groups of $5, 15$, and 35 drivers, respectively (with 30 minutes of training-data and 4 minutes of prediction-data). Moreover, experiments with variable amounts of training and prediction data we we showed that using this methodology any increase in amount of either the training or the prediction data monotonically improves the performance of the driver ID.

This architecture however has two major limitations *a*) sufficient amounts of data is needed for training (about 30 minutes) *b*) enrollment of a new driver requires retraining all models in OvR case, and training K new models in OvO case.. In the next chapter, we propose a method based on Gaussian mixture models that aims to address these limitations.

Your assumptions are your windows on the world. Scrub them off every once in a while, or the light won't come in.

Isaac Asimov

7 Driver Identification using Gaussian Mixtures

IN this chapter we present a driver ID method that can address some of the shortcomings of the method introduced in Chapter 6. The solution presented in the previous chapter only used signals available from the car's internal network (CAN-bus in this case). Recall that vehicle speed (VS) is one of the five signals we used in Chapter 6 for driver ID. Gao et al. [55] show that it is possible to infer the driver's route and destination only based on vehicle speed and the origin of a trip. For instance, an insurance provider most probably has the home or work address of their customer on file. An adversary in possession of this information and the vehicle speed can construct a profile of points of interest (POIs) visited by the driver. This may not be an issue as the service provider could have access to the location data from the car, but many do not want to share any information more than it is required. Therefore we decrease the number of signals used for driver ID from five down to two and focus only on the controls operated directly by the driver. And as a result also reduce the computational complexity.

In order to solve the scalability issues of the previous approach, we need a solution that the enrollment of new drivers has constant complexity. In other words, the cost of enrolling new drivers should not depend on the number of drivers currently enrolled in the system. Therefore, we propose to use Gaussian mixture models (GMMs) instead of discriminative classifiers. In this approach, the enrollment of new drivers does not require retraining of the previous driver models. As a result, the methodology introduced in this chapter is scalable. We will also see that we can obtain superior identification accuracy using a smaller amount of training and prediction data.

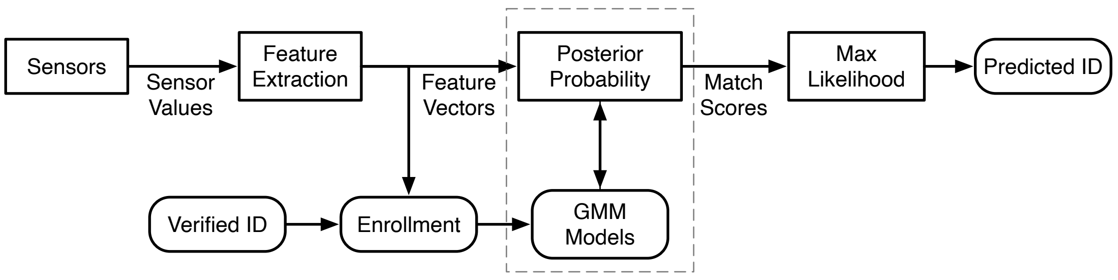


FIGURE 7.1: Method Architecture

The remainder of the chapter is organized as follows: In Section 7.1, we describe the proposed methodology and evaluation strategy. Section 7.2 presents a brief feature analysis. In Section 7.3, we present experiments and their corresponding results. Section 7.4 discusses the methodology. Lastly, in Section 7.5, we summarize and present future directions.

7.1 Methodology

In this section, we describe the proposed methodology, which phases are depicted in Figure 7.1. First we present the GMM for driver identification, and model fusion mechanism then we go over the feature extraction and analysis, and in the end the evaluation strategy.

7.1.1 The Gaussian Mixture Driver Model

The goal of driver identification is to associate a driving to its corresponding driver. We also assume that the set of target drivers is finite and that we have information about all candidate drivers. We propose to use GMMs for driver identification as they are well studied in the literature, in particular for speaker recognition [123, 132, 133].

Gaussian mixture model (GMM) denoted by λ is characterized by its probability density function, which is a weighted sum of M component densities:

$$p\left(\boldsymbol{x}|\lambda\right) = \sum_{i=1}^{M} \phi_i \mathcal{N}_i\left(\boldsymbol{x}\right) \tag{7.1}$$

where $\boldsymbol{x}$ is a d-dimensional feature vector, $\mathcal{N}_i(\boldsymbol{x}), i = 1, \cdots, M$, are the component densities and $\phi_i, i = 1, \cdots, M$, are the mixture weights. Each component density is a d-variate Gaussian of the form (Equation 7.2):

$$\mathcal{N}_i(\boldsymbol{x}) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)'\Sigma_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right\} \tag{7.2}$$

where $\boldsymbol{\mu}_i$ is the mean vector and $\boldsymbol{\Sigma}_i$ is the covariance matrix. The mixture weights should satisfy the constraint $\sum_{i=1}^{M} p_i = 1$. To train a GMM we need to estimate the following parameters:

$$\lambda = \{\phi_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\} \quad i = 1, \cdots, M. \tag{7.3}$$

Therefore each $\mathcal{C}_k$ driver is represented by a GMM and is referred to by their model λ_k.

7.1.2 Driver Identification

For driver identification, a group of K drivers is represented by K GMMs, $\lambda_1, \lambda_2, \cdots, \lambda_K$. Here our goal is to find the model which has the maximum a posteriori probability for a given set of observations (in our case observations are features extracted from each frame of driving data).

$$\hat{y} = \underset{1 \le k \le K}{\arg\max}\, P_r(\lambda_k | \boldsymbol{X}) = \underset{1 \le k \le K}{\arg\max}\, \frac{p(\boldsymbol{X}|\lambda_k) P_r(\lambda_k)}{p(\boldsymbol{X})} \tag{7.4}$$

where $\boldsymbol{X}$ is vector of observations at prediction time ($\boldsymbol{X} = \{\boldsymbol{x}_t\}_{i=1}^{T}$). Assuming that all the candidate drivers are equally likely to be the actual driver ($P_r(\lambda_k) = 1/K$) and taking into consideration that $p(\boldsymbol{X})$ is the same for all driver models, Equation 7.4 reduces to:

$$\hat{y} = \underset{1 \le k \le K}{\arg\max}\, p(\boldsymbol{X}|\lambda_k) \tag{7.5}$$

Assuming independence between measurements and using log to facilitate computations and improve numerical stability, we have:

$$\hat{y} = \underset{1 \le k \le K}{\arg\max} \sum_{t=1}^{T} \log p(\boldsymbol{x}_t|\lambda_k) \tag{7.6}$$

where $p(\boldsymbol{x}_t|\lambda_k)$ is defined by Equation 7.1.

7.1.3 Model Fusion

We consider two signals, percentage gas pedal (PGP) and SWA. We choose these signals because they are: 1) directly operated by driver 2) available through car's internal network (CAN-bus). Since these two signals come from two controllers that are operated separately, we model each with a GMM. We use the notation λ_G to refer to gas pedal operation model and λ_S for steering operation model. Moreover, because the two resulting models (λ_G and λ_S) are not equally informative, we need to give a higher weight to the model that is a better predictor of the driver. We use a parameter called α to indicate the weight that we associate to each model. Then at identification time, we combine their log-likelihoods by the ratio that is controlled by the parameter α. In this case, the

Equation 7.6 becomes as below:

$$\hat{y} = \underset{1 \leq k \leq S}{\arg\max}\{\alpha \log p(\boldsymbol{X}_G|\lambda_{G,k}) + (1-\alpha)\log p(\boldsymbol{X}_S|\lambda_{S,k})\} \tag{7.7}$$

where $\boldsymbol{X}_G, \boldsymbol{X}_S$ refer to gas pedal operation feature vectors and steering operation feature vectors respectively, and $\lambda_{G,k}, \lambda_{S,k}$ refer to the kth driver's gas pedal and steering models. The idea is that using the right weight will improve the accuracy. We propose to find the optimal α empirically in Section 7.3. Moreover, this separation will allow us to selectively take into account only one of the two signals, or changing the ratio in which we combine the log-likelihoods (α).

7.1.4 Evaluation Method

There are two important points we take into account in evaluating our method. Firstly, since we perform identification in small sets of K drivers (i.e., 5, 15, 35), it is important to account for unwanted side effects. For example, it could be that one set of 5 drivers have very distinct driving styles and therefore be easily discriminated, while another set of 5 drivers have similar driving styles and be difficult to discriminate. To prevent biasing our results with such phenomena we repeat each experiment with 30 random samples of K drivers. For each evaluation in order to use the whole data-set for both training and testing, we employ a cross-validation approach. In particular hold-one-out cross-validation, in which we first segment each trip into 10 slices. At each fold, we hold-out one slice from each of selected drivers, train the models, and test on the held-out slice. We use accuracy as the evaluation metric, as defined below:

$$acc = \frac{1}{k}\sum_{i=1}^{k} \mathbb{1}(y_i = \hat{y}_i) \tag{7.8}$$

where $\mathbb{1}$ is the indicator function, $\hat{y}_i$ and y_i are respectively predictions and true driver for i^{th} driving trace.
The overall accuracy score for an experiment is the average of all cases as follows:

$$acc_{score} = \frac{1}{R \cdot L}\sum_{r=1}^{R}\sum_{l=1}^{L} acc_{r,l} \tag{7.9}$$

where $L = 10$ is number of cross-validation folds and $R = 30$ is number of repetitions. In the rest of the work, we refer to this score as accuracy.

In our experiments we use notions of *training window* and *decision window*, the former refers to the amount of data used for training and the latter refers to the amount of data used for testing. Earlier we mentioned that we slice each driver's trip into 10 slices, and at each cross-validation fold 9 slices are considered to be part of training-set and 1

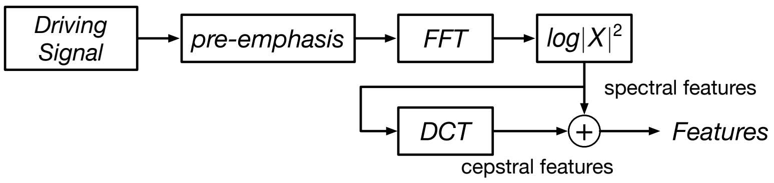


FIGURE 7.2: Feature Extraction Process

slice part of the test-set. In order to perform various analysis sometimes we truncate the train-set and test-set, for example, training window of 10 minutes and decision window of 1 minute means that from the training-set we use the first 10 minutes of the trip for fitting the models and from the test-set we use the first 60 seconds for predicting the driver.

7.1.5 Feature Extraction

A sliding window approach is used for feature extraction. We use a 2 seconds long Hamming-window and stride of 0.25 s. As it is shown in Figure 7.2 signal from each window frame is pre-emphasized[1] to enhance higher frequency components, we compute log-magnitude-spectrum of the signal. From here we perform two different set of operations to obtain two groups of features, spectral features and cepstral features.

For spectral features we consider log-power-magnitude of the signal for each 1Hz range to be a feature, up to 8Hz however, because higher frequencies have smaller magnitude we compress them in larger ranges in this case, 8–12Hz and, 12 Hz and above. We also include the sum of all energies as a separate feature, therefore we would have 11 spectral features. To obtain cepstral features we apply DCT to log-power-magnitude of signal and mean normalize it. In this work, we only keep the first 8 cepstral coefficients as features. Like spectral features, we also consider the sum of energy among the cepstral coefficients to be an extra feature as well. This will leave us with 9 cepstral features. This results in a feature vector of dimension 20 per signal. We also scale features using the following formula:

$$X_{scaled} = \frac{X_{raw} - \bar{X}_{raw}}{std(X_{raw}) + \epsilon} \tag{7.10}$$

to have each feature with a mean of close to zero and variance of one. ϵ is a very small machine-dependent value added to avoid division by zero. In our experiments, we scale the data only based on the training data, and apply the same scale to the test data.

7.2 Feature Analysis

In the initial experiments, we considered adding Δ and $\Delta\Delta$ of both cepstral and spectral features. Delta cepstral features are well studied in speech recognition field. They have

[1]$y\,[n] = x\,[n] - \beta x\,[n-1]$

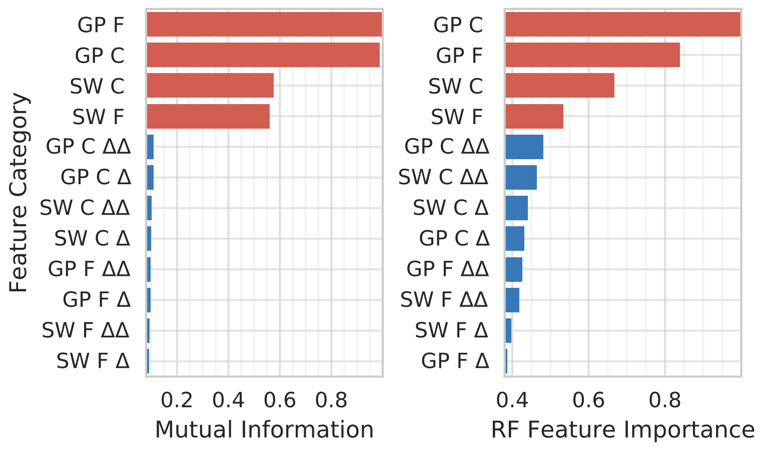


FIGURE 7.3: Normalized feature importance (larger the better), per feature category. GP stands for gas pedal and SW for steering wheel. F refers to spectral features, and C to cepstral features. No matter which notion of feature importance we use Δ or $\Delta\Delta$ features, are not informative. It is also evident that gas pedal features are more important.

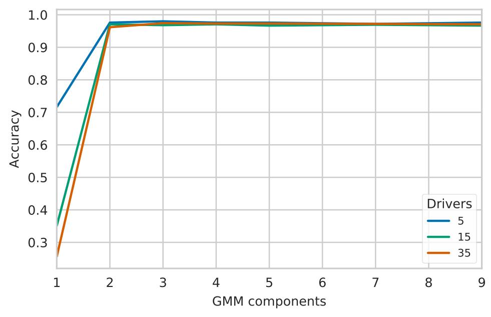
shown to improve accuracy by adding dynamic information to cepstral coefficients which in turn helps explain temporal dependency between the frames [134, 135]. Addition of Δ and $\Delta\Delta$ will increase the number of features to 60 per signal. Here we perform analysis to quantify and validate positive contribution of each of our signals and feature categories. We use two measures, mutual information and random forest feature importance, for both of which implementations from scikit-learn were used[86]. Results are presented in Figure 7.3. We can observe that both signals are important however, it is clear that percentage gas pedal (GP) presents higher importance. When it comes to comparing feature categories, cepstral features show a slight advantage however the gap is not significant. It is also clear that δ features do not play an important role and have low importance. We also validated this in our preliminary results as we would usually get better results without δ features. As a result we decided to remove them altogether for the sake of simplicity and performance.

7.3 Experiments

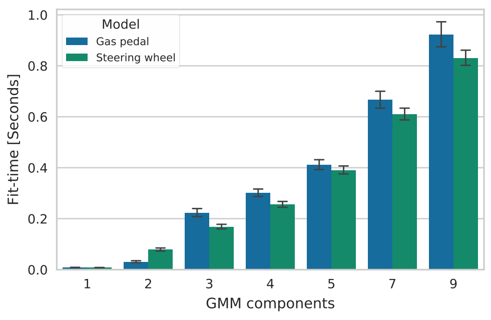
In this section, we will evaluate our proposed methodology from various aspects and analyze how model parameters affect the performance.

7.3.1 Model Parameters

We seek optimal values for two parameters M which indicated the number of GMM components and α which is the ratio by which we linearly combine the likelihoods from steering and gas pedal models. First, to determine the optimal number of GMM components we perform a set of experiments by varying number of GMM components. We



(A)



(B)

FIGURE 7.4: *(A)* When number of Gaussian components crosses 2, there is little effect on the accuracy of the model. When looking more closely, there is a slight improvement for 3 components, but after that accuracy plateaus. *(B)* We also measure the time needed to fit a GMM (with full covariance matrix). We can see that fitting time linearly increases with the number of components.

consider driver models with 1 to 9 Gaussian components and fixed $\alpha = 0.5$, and evaluate their performance. From the results presented in Figure 7.4a we can see that accuracy plateaus starting from 2 GMM components and the best result is obtained with 3 GMM components.

Depending on the application some time-constraints maybe need to be satisfied by the driver identification solution (e.g., detecting fraud, personalizing car configurations per driver). Therefore we also investigate the training time of the models. Figure 7.4b shows the obtained results. As one would expect the fitting time increases with the number of GMM components. So for each driver total training time would be the sum of gas pedal (λ_G) and steering (λ_S) models. Similarly, for a scenario with 5 drivers, the total training time is going to be approximately 5 times the fit time of a single driver. Since with $M = 3$ the required computational time to train the models is still reasonable in the following sections we always use models with 3 GMM components.

Second, to obtain the optimal combination of gas pedal and steering features, we vary α from 0 to 1 and evaluate our method. Having $\alpha = 0$ is equivalent to only using λ_S and $\alpha = 1$ would be equivalent to just using λ_G. This is done very similar to [35], however here we are using different signals and features. The corresponding results are presented in Figure 7.5. As it can be observed, the way α affects the results varies with the number of drivers. It is clear that as the number of drivers goes up λ_S has a more positive effect on the results, this is in line with the findings in our previous work [106].

Drivers operate the gas pedal and the steering wheel only at parts of their drive. For instance, when the road is straight, no steering is required, and there are times that driver lifts their feet from the pedal either to slow down or to keep a constant speed; At such moments data from the corresponding signals is not quite informative. Therefore we also considered dynamically switching between the models (λ_G and λ_S) based on

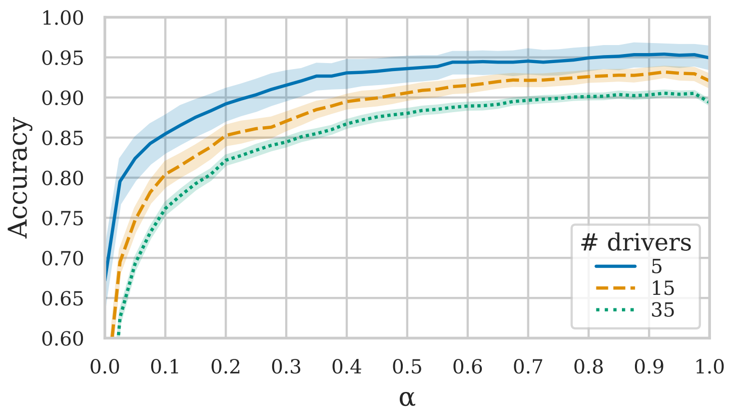
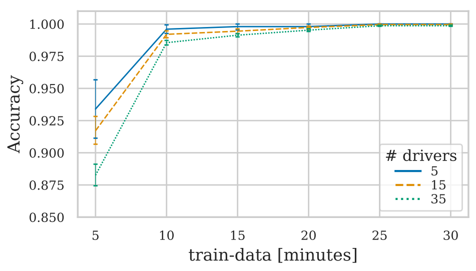
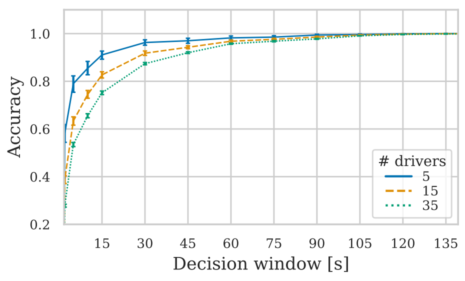


FIGURE 7.5: The impact of α on accuracy. α close to 1 favors the PGP, which also results in better performance. However close to 0.9 the improvement plateaus and we can see a drop in accuracy, close to 1.0

activity in signals (average energy) however we did not observe significant improvements in performance.



(A)



(B)

FIGURE 7.6: The impact of the amount of training and prediction data on identification performance. *(A)* We can see that even with five minutes of training data we obtain close to 90% accuracy, and with 10 minutes, we get above 97.5% accuracy. *(B)* Accuracy quickly improves with larger decision windows. 45 s prediction data is needed to obtain accuracy of above 90%.

7.3.2 Sensitivity Analysis of Training-data

One of the important factors in driver identification is driver enrollment. In order to add a new driver to the system, we need to collect sufficient amount of data from the driver to fit a model. To find out how much data is enough we perform experiments by varying the amount of train-data in 5-minute steps from 5 to 35 minutes of driving data and measure the performance with a fixed 2 minutes of test-data. The results are presented in Figure 7.6a. As one can see 5 minutes of training data yields poor results, however starting from 10 minutes and above, performance crosses 90%. From this plot, we can conclude that with just 10 minutes of driving data we can obtain good accuracy and as we collect more data from the driver it is possible to improve the model [136].

7.3.3 Sensitivity Analysis of Decision Window

Driving is a repetitive task, however, it takes a few minutes until one repeats various maneuvers during a driving session. For example, consider driving on a straight stretch of road, there is no steering maneuvers and only a few pedal operations. Therefore to obtain more reliable results, a larger decision window is required. Moreover, in applications such as theft detection and comfort, you would want to be able to accurately identify the driver in the least time possible. Therefore we evaluate the impact of test-data on performance. To do so we run experiments with a fixed training data of 20 minutes but with variable sizes of decision window. We cover variations from 30 seconds to 5 minutes. A 60 seconds long decision window leads to good results but starting from 90 seconds for all scenarios we obtain more than 90% accuracy.

7.3.4 Overall Identification Performance

Here we present a summary of the results obtained in our experiments. The results are presented in Table 7.1. We selected two cases, one with 15 minutes of training data to fit the models representing a use case that not much of data from drivers is available and one with 30 minutes of training data, representing a case with higher data availability. For each case then we present accuracies for three decision window lengths, 1, 2 and 4 minutes. The choices are made to cover a wide range of applications and accuracies we can get with our solutions under these varying conditions. We can see that even with 15 minutes of training data, after two minutes we can reach accuracies of over 95% for 5 and 15 drivers and 94% for 35 drivers. We also include extended experiments with 50 and 67 (the whole data-set). We can observe that with further increase in number of drivers accuracy slightly decreases, however, this to large degree can be compensated with collection of more data both for training and for prediction.

7.4 Discussion

Table 7.1 compares the identification accuracy of the GMM with the method presented in the previous chapter. We improved the results significantly, primarily when a smaller amount of training data is used. For instance, for the case with ten minutes of training and three minutes of prediction data, the accuracy increase by 26.26, 42.32, 57.85 percent for 5, 15, 35 drivers, respectively.

Moreover, we reduced the number of signals used down to two and used only 20 features per signal. Another significant improvement over the previous method is the

TABLE 7.1: Comparison of the GMM with Discriminative approach.

Train-data	Decision	Accuracy - Mean (Std.)					
	Window	5 Drivers		15 Drivers		35 Drivers	
Minutes	Minutes	Discrimi.	GMM	Discrimi.	GMM	Discrimi.	GMM
5	1	0.575 (0.106)	0.883 (0.068)	0.438 (0.061)	0.853 (0.027)	0.367 (0.032)	0.802 (0.018)
	2	0.589 (0.109)	0.909 (0.075)	0.470 (0.062)	0.894 (0.029)	0.415 (0.035)	0.863 (0.020)
	3	0.594 (0.107)	0.918 (0.070)	0.494 (0.061)	0.906 (0.031)	0.431 (0.035)	0.876 (0.021)
	4	0.597 (0.116)	0.933 (0.062)	0.509 (0.063)	0.913 (0.030)	0.439 (0.038)	0.879 (0.023)
10	1	0.743 (0.091)	0.953 (0.039)	0.617 (0.044)	0.929 (0.020)	0.537 (0.030)	0.903 (0.011)
	2	0.775 (0.100)	0.989 (0.017)	0.674 (0.053)	0.978 (0.012)	0.600 (0.028)	0.971 (0.007)
	3	0.788 (0.096)	0.995 (0.010)	0.697 (0.061)	0.992 (0.007)	0.624 (0.025)	0.985 (0.004)
	4	0.795 (0.101)	0.997 (0.008)	0.714 (0.051)	0.992 (0.007)	0.640 (0.024)	0.985 (0.005)
20	1	0.905 (0.058)	0.975 (0.025)	0.796 (0.047)	0.956 (0.016)	0.721 (0.021)	0.939 (0.009)
	2	0.921 (0.055)	0.995 (0.010)	0.857 (0.038)	0.987 (0.010)	0.790 (0.023)	0.985 (0.005)
	3	0.929 (0.054)	0.999 (0.005)	0.888 (0.032)	0.997 (0.004)	0.825 (0.021)	0.994 (0.003)
	4	0.943 (0.048)	0.998 (0.006)	0.908 (0.031)	0.998 (0.004)	0.851 (0.020)	0.995 (0.003)

fact that the addition of new drivers to the system is as easy as collecting training data and fitting two GMMs. In the previous work, we had to retrain every time we enroll a new driver in the system.

This method is also more privacy-preserving because only using the speed and knowing the workplace or residential address of a person one could infer their destination using only the speed [55]. Therefore with this method, we cannot learn anything more than pedal or steering wheel operation patterns, which do not reveal much information about a person's lifestyle.

We examined several ideas that did not improve the results. One of these ideas is the *universal background model* (UBM) [133]. In the context of GMMs, the idea is that first, we compute a model (*UBM*) for the general population, then for each individual, we adapt the GMM components to fit the distribution of the feature vectors of a specific driver. This has been shown effective in speaker recognition, but in our experiments, it did not improve the results for the driver ID task. Since the idea is sound we suspect the reason may be a lack of sufficient training data to construct a *UBM*.

7.5 Summary

In this chapter, we proposed a driver ID methodology that delivers excellent accuracy while being computationally light-weight. We only used two signals (gas pedal position and steering wheel angle) that are easily accessible from CAN-bus. Inspired by the speech recognition research, we extracted cepstral and spectral features using a sliding window mechanism. The resulting feature vectors were used to fit two GMM for each driver (one per signal). At prediction time, feature vectors are extracted from the driving data, and the driver is predicted based on the maximum likelihood estimation principles. We linearly combine likelihoods from two models, for which we obtain the optimum ratios (α) empirically. We provided analysis covering variations in model parameters, training, and prediction conditions. We showed that with 30 minutes for driving data for training and 4 minutes of driving data for prediction, our proposed method achieves an accuracy of over 99% for scenarios with 5, 15, and 35 drivers. This method has several advantages. It is computationally light-weight. It is privacy friendly, because it does not depend on data such as speed, which may lead to compromising the routes taken by the driver [55]. Moreover, this approach is scalable because the enrollment of new drivers does not require retraining of other models, which was the case for the method proposed in the previous chapter and the majority of the related work.

For future work, we propose investigations on the portability of the GMM models between the vehicles. Unfortunately, in our dataset all participants drive the same car. However, it would be of interest to see if our proposed method will also work if the driver changes the cars. This would be of interest to logistics companies when the driver is not tied to a single vehicle or when an insurance customer buys a new car; ideally we should be able to use the same model across various vehicles.

In the next chapter, we present a deep learning-based approach that allows us to achieve excellent accuracy with little data.

Who then is invincible? The one who cannot be upset by anything outside their reasoned choice.

Epicetus, Discourses, 1.18.21

8 Driver Embedding and Its Applications to Driver Identification

IN the past two chapters, we approached the driver ID as a classification problem. We showed that the discriminative classifiers limit the scalability, and require more than 20 minutes of training data to show good accuracy. On the other hand, the GMM model solves some of the issues with the discriminative models and shows excellent accuracy. However, still, we need about 5–10 minutes of driving data to be able to estimate a good model that can accurately identify the driver. There are use-cases that require to obtain high accuracy with little data, let us take the example of a car rental service. Since they charge extra fee for each additional driver. They would like to construct a model before the driver leaves their premises. This would mean that we need to construct a model with less that a minute of driving data.

Recently, DL has shown a great promise in several areas of research. The main issue with DL methods is the need for substantial amounts of training data. To reduce the amount of data needed to construct a driver model, we propose a similarity learning approach. We briefly outline the general idea, which we further expand and describe in detail in this chapter. In this approach we train a DNN to encode an interval of driving data into an embedding. Embeddings are n-dimensional real-valued vectors, that often have some useful properties. They are often used as a compact method of representing a complex concepts. For example, in natural language processing (NLP), words are represented in form of embeddings, which allows the DL methods to capture the meaning and semantics of the words.

In our approach, we expect the embeddings from the same driver to be in a close Euclidean distance from each other. On the other hand, we expect the embeddings from different drivers to be far from each other. We can use this property to develop a driver ID method. The significant advantage of this approach is that we do not need to train the network using the set of drivers that we are interested in identifying. Therefore, we can use any large driving dataset to train the network. Then, we feed driving data from our target set of drivers to this pre-trained DNN. The resulting embeddings serve as a compact driver representation, or a driver fingerprint. We use these embeddings in conjunction with a k-NN classifier to construct a driver ID method.

Many have used the embeddings in contexts other than text and language processing. There are a few works that have done the same in driving and mobility context. Hallac et al. [137] propose a method they call *Car2Vec* which uses embeddings to represent state of the car. Their focus is on compressing large number of sensor values (665) down to a 64-dimensional real valued vector. Liu et al. [138] propose a method based on deep sparse auto-encoder to visualize driving data. Liu et al. [139] apply the *Word2Vec* [21] to road network to obtain road embeddings that are used to quantify complex traffic interactions among roads and capture underlying heterogeneous and non-linear properties. In a more relevant work, the authors propose to use a Siamese network [140, 141] in conjunction with an *embedding net* based on stacked LSTMs. They only operate on left turn events. They validate their approach on a dataset of 32 drivers, each time 30 drivers are used for training and two for testing. They obtain ROC area under the curve (AUC) of 0.753 in average Dang and Fürnkranz [142].

The rest of the chapter is organized as follows. First, in Section 8.1, we introduce the *deep driver* model and the details of our approach. In Section 8.2, we describe the evaluation setup. Then in Section 8.3, we present the results of the driver ID and verification experiments. In Section 8.4, we visualize some of the resulting embeddings. Next, in Section 8.5, we discuss some of the related works and potential applications of the proposed method. Lastly, in Section 8.6, we summarize the work.

8.1 Methodology

In this section, we describe the proposed methodology. We start by presenting the Triplet network architecture and the training process. Then we describe the data and the embedding network, which is a deep residual neural network.

8.1.1 Deep Driver Model

Deep learning models have been successful at various classification tasks, such as image classification. Although there have been attempts to interpret and utilize intermediate

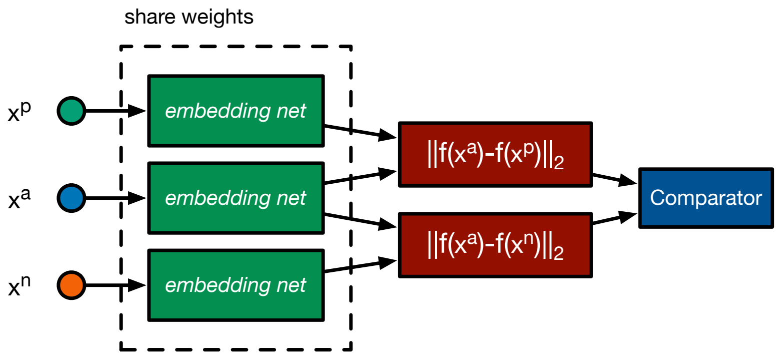


FIGURE 8.1: Block diagram of Triplet network. Three duplicates of *Embedding net* with shared weights encode each triplet examples. We compute the L_2 distance between the positive and the negative pairs. We optimize the network in order to reduce the distance between the positive pair and increase the distance between the negative pairs.

representations in deep networks, these intermediary representations were only a side effect of the main task of the deep network (i.e., image classification).

Starting with variants of *Siamese Network* that used a contrastive loss over the representations used to discriminate between the pairs of examples perform tasks such as signature or face verification [140, 143, 144]. Contrastive loss promotes a small distance between pairs of the same label and a large distance when the labels are not the same. Siamese networks are sensitive to calibration and are not suitable for classification [52]. We use a *Triplet network*, which was introduced by Hoffer and Ailon [52] as an alternative for the Siamese network with better performance.

Before we describe the Triplet network, we determine the input data to use for the driver ID task. The input to a face or signature verification system is trivial, it is the image of a face or a signature. However, for driver ID, we face an endless stream of sensor measurements, with no particular pattern. There are several ways to approach this problem, such as using specific events as the input, for example, the interval of data corresponding to left-turns, or harsh brakes. However, in short journeys, such events may not occur frequently; moreover, isolating those events requires extra preprocessing. For practical purposes, we consider the input to be a short interval of driving data that may or may not correspond to any driving event. We refer to this interval as a *maneuver*, and for simplicity, denote it by x.

In Triplet network architecture the goal is to find an embedding $f(x)$ that maps the input x into a space $\mathbb{R}^d$ in a way that L_2-norm of the distance between all the maneuvers of the same driver is small, while the L_2-norm of the distance between pairs of maneuvers from different drivers is large.

The embedding is represented by $f(x) \in \mathbb{R}^d$, which embeds a maneuver x into a d-dimensional Euclidean space. We want to ensure that a maneuver x_i^a (*anchor*) of a particular driver is closer to all other maneuvers x_i^p (*positive*) of the same driver that it is to any maneuver x_i^n (*negative*) of any other person (see Figure 8.2). We formalize this

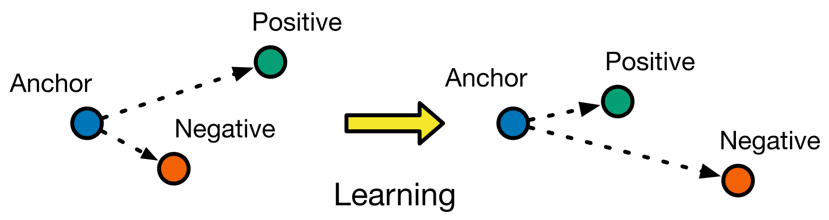


FIGURE 8.2: Left, an exemplary triplet is shown, the distance between the anchor (blue) to negative example (orange) may be smaller than the distance to the positive example (green). After the learning process, the distance of the anchor to the positive example is smaller than the distance to the negative example.

as:

$$||f(x_i^a) - f(x_i^p)||_2^2 + \alpha < ||f(x_i^a) - f(x_i^n)||_2^2, \forall (x_i^a, x_i^p, x_i^n) \in \mathbb{P}. \tag{8.1}$$

where α is a margin between the negative and positive pairs. $\mathbb{P}$ is the set of all possible Triplets in the training set and has the cardinality of N. Equation 8.1 is also referred to as the *Triplet loss*. We call this adaptation of the Triplet network architecture, which maps the driving data to an embedding space, the *deep driver* model. Figure 8.1 depicts a generic Triplet network. The main building block of the Triplet network is the *embedding network*. The *embedding network* is a DNN and its architecture depends on the input data. For simplicity, we defer presenting the embedding network architecture to the end of this section.

8.1.2 Training

The training of a Triplet network architecture is nuanced. Depending on the number of examples in the training dataset, the number of possible triplets will be very large. Iterating over all the possible Triplets will take too long. Moreover, it is not efficient because many of Triplets already satisfy Equation 8.1, and therefore iterating them will not improve the model. To converge faster, we should select the Triplets that violate the constraint in Equation 8.1. There are two main approaches to achieve this goal:

- **Offline:** To generate the Triplets periodically after n mini-batches, and find the examples that violate the constraint the most.
- **Online:** To generate the Triplets online by selecting the hard examples from a mini-batch.

Initially, we performed experiments with both approaches, but we decided to generate examples online for each mini-batch. Because pre generating all Triplet for the whole dataset is slow and time-consuming, moreover selecting the globally hardest examples may lead to too much focus on the erroneous or poor quality examples. In our case, this could be where the car is not moving, and there is no pedal operation; therefore, no information to discriminate drivers.

The final process is as follows, for each mini-batch, we construct all Triplets consisting of all possible combinations of anchor-positive pairs. However, for the negative pair, we do not choose the hardest example (i.e., negative example closest to the anchor) because Schroff et al. [145] found that, selecting the hardest negatives can quickly lead to a local minima and result in a collapsed model, $f(x) = 0$, something that we also experienced in the early experiments. Therefore to avoid a collapsed model, we select a *semi-hard* example, defined by:

$$||f(x_i^a) - f(x_i^p)||_2^2 < ||f(x_i^a) - f(x_i^n)||_2^2 \tag{8.2}$$

We use the Adam optimizer [146] with *learning_ rate* of 0.005, a learning rate scheduler halves the learning rate every 20 epochs. We continue the training for 70 epochs and perform an early stopping if the loss does not decrease for ten consecutive epochs. The training is done using a Tesla V100 GPU, and the implementations are done in PyTorch [147]. At the training time, we train five models with different random initializations and at the end, proceed with the best model.

8.1.3 Input Data

We use the Uyanik dataset for validation, two signals are selected for feature extraction, steering wheel angle (SWA), and percentage gas pedal (PGP) signals. These are the minimal signals that are usually available to obtain from the car's internal network. We used the same signals in Chapter 7 and established their effectiveness. By re-using the same signals, we focus on the new approach instead of the intricacies of signal and feature selection.

From each signal, we extract the same features as used in Chapter 7. A sliding window approach is used for feature extraction. We use a 2 seconds long Hamming-window and stride of 0.25 s. First, we pre-emphasize[1] the signal to enhance higher frequency components. Then, to obtain spectral feature we compute the log-magnitude-spectrum of the signal. For spectral features, we consider log-power-magnitude of the signal for each, 1Hz range to be a feature, up to 8Hz; however, because higher frequencies have a smaller magnitude, we compress them in larger ranges in this case, 8–12Hz and, 12Hz and above. We also include the sum of all energies as a separate feature, which amounts to 11 spectral features. To obtain cepstral features, we take the DCT of log-power-magnitude of signal and mean normalize it. In this work, we only keep the first eight cepstral coefficients as features. Similar to the spectral features, we also consider the sum of energy among the cepstral coefficients as an extra feature. This will leave us with nine cepstral features. This results in a feature vector of dimension 20 per signal.

[1] $y[n] = x[n] - \beta x[n-1]$

To summarize, a frame consists of several consecutive sensor measurements. The features are extracted per frame. The frames advance in time in strides of 0.25 s. Therefore a feature vector of dimension 40 (20 per signal) is extracted every 0.25 seconds (*stride*=0.25s). We call n consecutive feature vectors a *block*. We use $n = 32$, therefore a block $X \in \mathbb{R}^{40\times32}$, which covers 9.75 s of driving data. A *block* is the unit of data that we input to the embedding network. Although a *block* can be thought of as a two-dimensional image, it is a 40-dimensional time-series data of length 32, therefore the neural network should treat it appropriately.

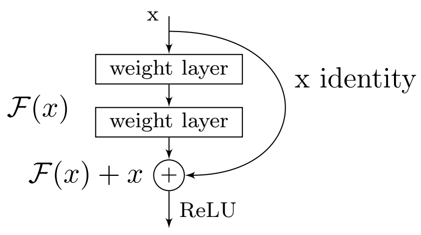


FIGURE 8.3: A building block of residual learning

8.1.4 Embedding Network

We need to choose a deep neural network architecture that is suitable for the input data. Several architectures exist for the common tasks such as image classification [25, 26], and language modeling [23, 148]. Unfortunately, not only there is no established architecture for driving data, there are few works focused on temporal sensor data (d-variate time-series). We propose a residual architecture inspired by ResNet [27], a well-known architecture for image classification. The main difference is that we only use three residual blocks and *1*-dimensional convolutions instead of *2*-dimensional convolutions.

Residual networks were introduced to solve the training accuracy degradation in very deep networks. Residual networks are composed of residual blocks that each have a *shortcut connection* that skips through from the beginning of the block and is added to the block's output. The idea is that if we hope that the block learns the mapping $\mathcal{H}(x)$ it can as well learn $\mathcal{F}(x) := \mathcal{H}(x) - x$. The hypothesis is that it is easier to learn $\mathcal{F}(x) + x$ than it is to learn $\mathcal{H}(x)$ itself (see Figure 8.3).

Each residual block consists of three *1*-dimensional convolutional layers. The first layer has a Kernel size of seven, the second one five, and the last one three. The number of the feature-maps (filters) is 64 in the first block and 128 in the last two residual blocks. Similar to the main ResNet paper, each convolutional layer is followed by a batch normalization (BN)[149] and a rectified linear unit (ReLU) non-linearity [150]. The residual blocks are followed by an average pooling layer and a fully connected (FC) layer. This architecture is depicted in Figure 8.4.

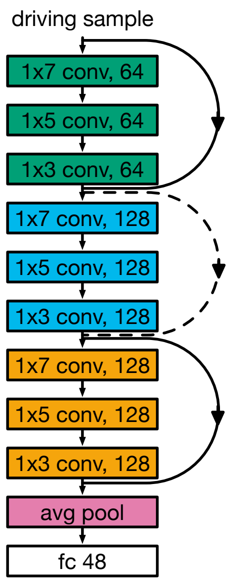


FIGURE 8.4: Simplified representation of the residual network used in this work as *embedding network*. Residual blocks can be distinguished by their color. Each residual block contains three *1*-dimensional convolutional layers.

To confirm the capability of this network in solving the driver ID problem, we perform an experiment. To use the *embedding network* as a classifier, we add a non-linearity (ReLU), followed by an FC layer with Softmax activation to the end of the network. We only test this case with the entire dataset of 54 drivers. We perform 10-fold cross-validation and split each trip into ten segments, and at each fold, we leave out one segment and train on the rest. The setup is generally similar to Chapters 6 and 7, except that there is no random sampling of the drivers. We make predictions for every block of data in the test set and make decisions based on the maximum sum of scores. The model achieves the accuracy of 0.9931 with standard deviation of 0.0272. Even though we performed this experiment to validate the efficacy of the architecture for modeling driving data, it outperforms our earlier approaches, but we should note that this setup is not scalable and requires complete retraining when we enroll a driver.

8.2 Evaluation

We evaluate the performance of the deep driver model for two tasks of driver ID and driver verification. We use the Uyanik dataset for the evaluations. Although there are slight variations the underlying setup is the same for both tasks.

8.2.1 Experimental Setup

There are 57 drivers in the Uyanik dataset. To obtain an accurate performance estimation, we repeat the experiments several times. We do this at two levels. First level is where we randomly sample drivers to use as test set. The second level is where we randomly sample intervals of driving data to use as driver model, and test example.

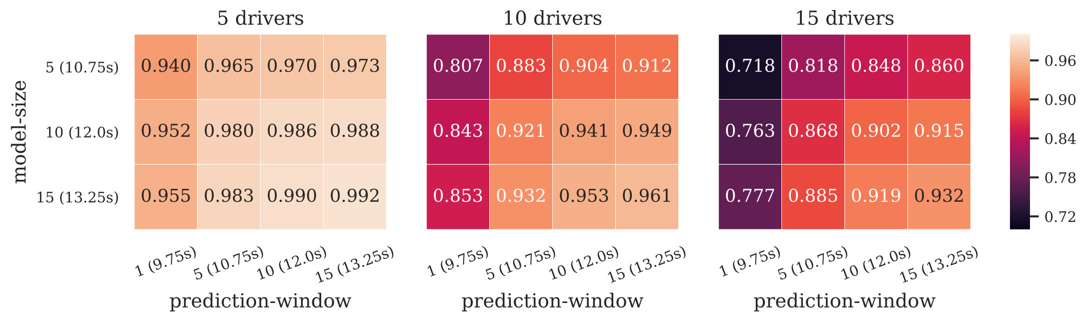
At each round, a set of K drivers ($K \in \{5, 10, 15\}$) is randomly sampled to take part in the evaluations. We repeat all experiments $R = 30$ times with different random subsets of K drivers. We consider these drivers as the validation set. Then we train the DNN using data from the remaining drivers. Therefore, when we evaluate a driver ID task of $K = 10$ drivers, $57 - 10 = 47$ drivers remain for the training of the DNN. In another example, when five drivers are selected in validation, $57 - 5 = 52$ drivers are left for training. Therefore, the model with ten drivers not only tackles a more difficult task (discriminating twice the number of drivers) it also is at a data disadvantage, because there is less data remaining for training.

The evaluation process is different from the previous chapters. To evaluate the performance of this approach, we design a procedure to generate test scenarios. Each scenario consists of sampling a *model* and L *positive examples* for each driver $\mathcal{C}_k$. We always make sure that there is no overlap between the *model* and the *positive-examples.* We repeat this M times, to achieve a fair evaluation of the models. A *model* in this case consits of one or more consecutive *blocks.* We introduce the variable *model-size*, to denote the number of *blocks* used to construct a driver model. For example, the *model-size* of five means that we use five consecutive *blocks* to construct a model. Here we used the term construct, because we do not necessarily fit a model using this data. A *positive-example*, refers to a sequence of consecutive *blocks*, used for prediction, the number of these *blocks* are determined by the variable *prediction-window.* The *model-size* and *prediction-window* are proxies for the amount of driving data used for the enrollment (*model-size*) and the prediction (*prediction-window*). We experiment with *model-size* $\in \{1, 5, 10, 15\}$ and *prediction-window* $\in \{5, 10, 15\}$, $M = 50$, $L = 50$. We use this procedure to generate datasets that we use in experiments presented in the next section.

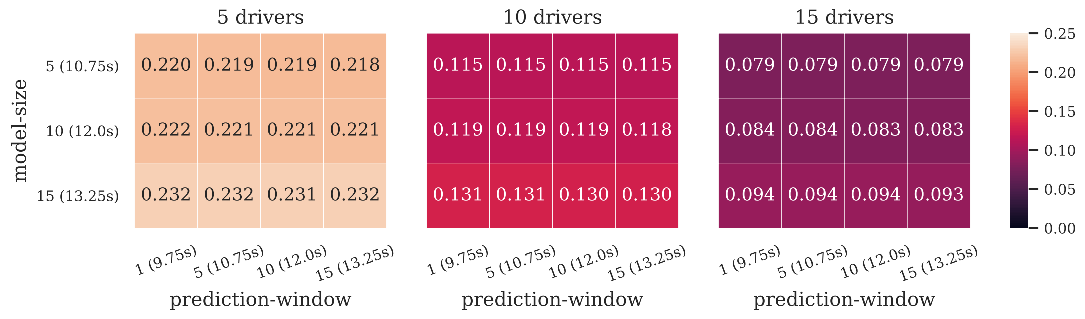
8.3 Experiments

8.3.1 Driver Identification

The deep driver model encodes a *block* into a d-dimensional *embedding.* In order to perform driver ID with such a mechanism, we need to extract embeddings from our training and test data. Then each resulting embedding can be treated as an example, therefore we can use any classifier such as k-NN [151] or SVM [91]. For our experiments, we use the procedure presented in Section 8.2 to generate examples. We evaluate the performance per scenario. We use the trained DNN to extract embeddings for all *models* and *positive examples.* Then we use the model *embeddings* for training and the embeddings of the *positive examples* for *evaluation.* We use the k-NN classifier because it is suitable for problems where only few training example are available (*model-size*). k-NN in order to



(A) Deep driver model



(B) Gaussian mixture model

FIGURE 8.5: *(A)*: Identification performance of Deep driver model, for 5, 10, and 15 drivers. We can see for all three cases, accuracy of over 0.93 achieved with less than 15 seconds of data. *(B)*: Results from a GMM model under similar conditions. The accuracy for all cases is only slightly better than random choice. GMM needs at least a few minutes of training data to produce high accuracy.

classify an unlabeled example finds the k closest examples in the training set and selects the majority class. Therefore driver ID works as follows, we use the *model*'s embeddings as labled examples to construct a classifier (*model-size* embeddings per driver). At prediction time, given a *positive example*'s embeddings, the classifier can make prediction per each embedding (depending on *prediction-window*) and final prediction is determined by majority vote. We use k-NN classifier from scikit-learn software package [86], with five neighbors and Euclidean distance metric. Since the same amount of examples are sampled per each class (driver) we choose accuracy as the chosen performance metric. We report the average accuracy accross scenarios and runs (experiments are repeated 30 times with random subsets of drivers).

This approach eliminates the need for retraining the DNN when new drivers are enrolled in the system. For example, when we experiment with 5 drivers, the DNN is trained with the data from all drivers except the 5 drivers selected for the ongoing experiment. Therefore, to enroll a new driver, the only requirement is to obtain a few blocks of driving data, which we feed to the DNN to obtain their embeddings. Then the embeddings are used by the k-NN classifier.

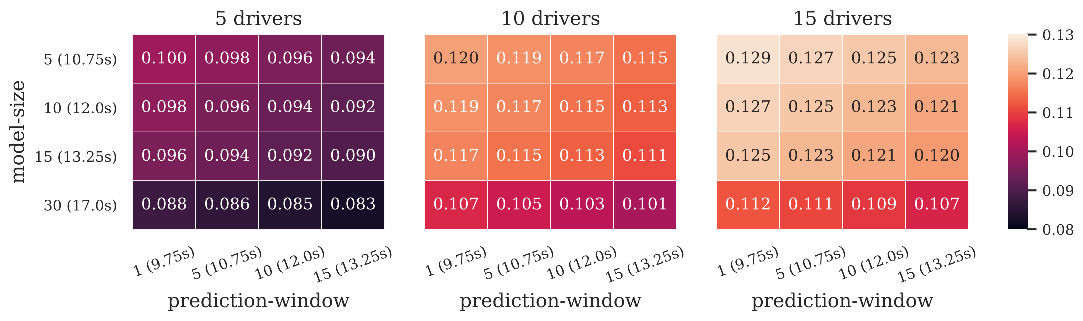


FIGURE 8.6: Deep driver model driver verification - EER

The results are presented in Figure 8.5a. The best results are achieved with *model-size* and *prediction-window* of 15 *blocks* which is equivalent to 13.25s with accuracy of 0.99, 0.96, and 0.93 for 5, 10, and 15 drivers, respectively. We observe that the accuracy increases proportional to *model-size* or *prediction-window*, which is expected.

We introduced the deep driver model to reduce the amount of training data required to achieve accurate driver ID. To confirm our hypothesis, we run experiments with the exact same setup used here to evaluate the deep driver model but instead we use the GMM model introduced in Chapter 7. Therefore, instead of the *blocks* of data and their corresponding *embeddings* we use the feature vectors extracted from frames that amount to same time duration, for example 13.25s in the case of *model-size* of 15. The corresponding results are shown in Figure 8.5b, for the case with *model-size* and *prediction-window* of 15, we only obtain the accuracy of 0.23, 0.13, and 0.093 for 5, 10, and 15 drivers, which is only slightly more that uniformly random prediction.

At first it appeared that the results incorrect, but complementary experiments showed that the only issue is the small amount of data used for fitting the GMMs (*model-size*), and larger *model-sizes* (in order of minutes) will result in high accuracy. An interesting observation here is that the increase in *model-size* slightly improves the accuracy but the increase in *prediction-window* does not improve the accuracy at all, we confirmed that even much larger *prediction-windows* did not improve the results. Since the data used for fitting the GMMs does not adequately represent the distribution of driver's feature vectors, no matter how much data is used in *prediction-window* the accuracy does not improve.

8.3.2 Driver Verification

Even though driver ID and verification are closely related tasks in the previous chapters we did not consider the driver verification task. Since the deep driver model is optimized as a distance metric, it is a ideal for the driver verification. We consider driver verification to be the task of confirming whether an example belongs to a certain driver or not.

The same procedure as described in Section 8.2 used to sample examples. Similar to driver ID, we extract embeddings from sampled *model* and *positive example blocks*. In the verification case, for *model-sizes* or *prediction-windows* of more than 1, we merge all their corresponding *embeddings* by averaging. Next, we need to produce *model-example* pairs. In each scenario, assuming K drivers and L examples per driver. We produce all the combinations of positive and negative pairs of model and examples. Each driver has L positive pairs and $(K-1)*L$ negative pairs. Then we obtain embeddings for each pair. Next, we compute the square of the L_2-norm of their difference.

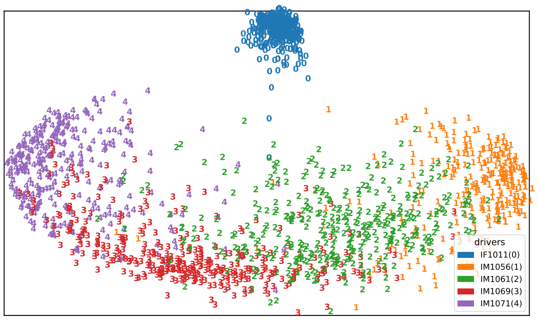
$$||f(x_i^{model}) - f(x_i^{example})||_2^2 \tag{8.3}$$

Based on this distance, a decision can be made to accept or reject the example. Since verification is a binary classification task one may set the decision threshold at any operating point, which will affect the false-positive and false-negatives. Depending on the application a multitude of methods can be used to select a decision threshold that reflects the designer's needs (such as methods that associate a higher loss to false-positives). We only report the EER, which is the value of the false rejection rate (FRR) or false acceptance rate (FAR) at the operating point in which both of them are equal. The results are shown in Figure 8.6. In the case of *model-size* 30 and *prediction-window* of 15—equivalent to 17 and 13.25 seconds of driving data, respectively—we obtain EER of 0.083, 0.101, and 0.107 for 5, 10, and 15 drivers.

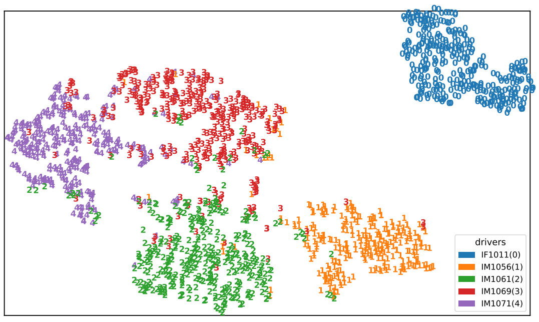
8.4 Embedding Visualization

The deep driver model encodes *blocks* of driving data into embeddings. The main advantage of the embeddings are their semantic meaning ([21, 139]). Even though the focus of this work is to identify or verify the driver, these embeddings can also be used for other purposes. We used the embedding dimension of 48, in order to visualize them we use two common dimensionality reduction methods, PCA and t-distributed stochastic neighbor embedding (t-SNE) [152] to project the embeddings onto two dimensions. We visualize a random sample of embeddings from four experiments involving five drivers. Figure 8.7 shows the corresponding results. Each row are the representations of the same embeddings, the figures on the left are projected using PCA and the ones on right are projected using t-SNE. Each point in the figure represents embedding of a *block* of driving data, the color and shape of each point varies for each driver.

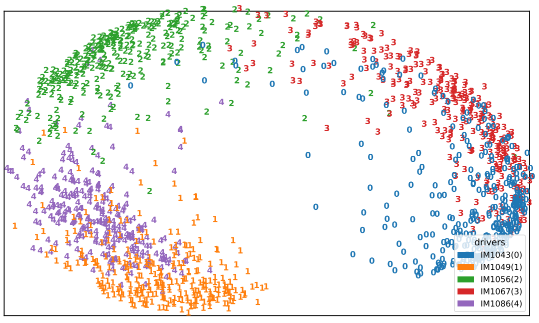
In most cases t-SNE projections show a clear separation between drivers. We did not handpick the examples with good separation on the contrary we tried to find some examples that are difficult to separate (at least using the PCA projections). This clear separation enables even more applications such as infering number of drivers that used



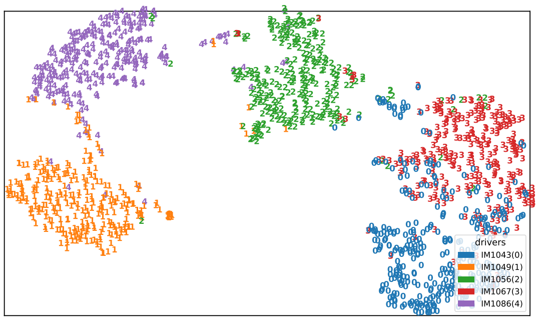
(A)



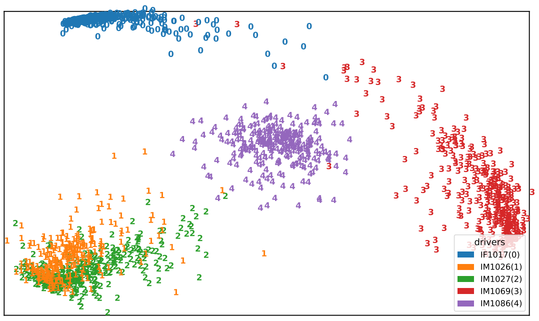
(B)



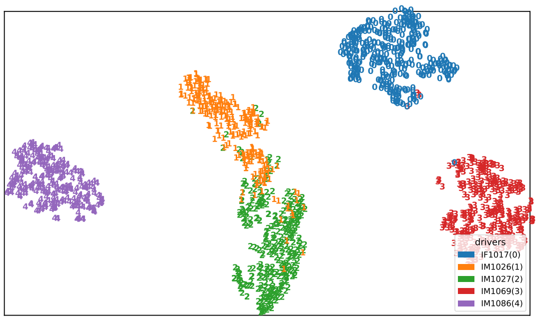
(C)



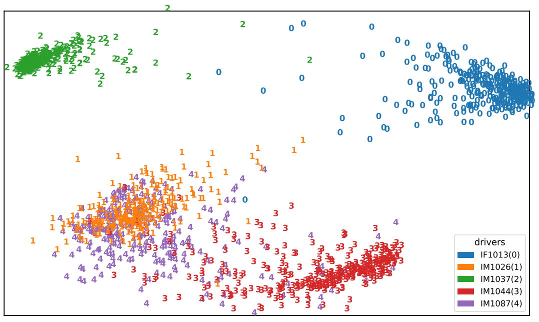
(D)



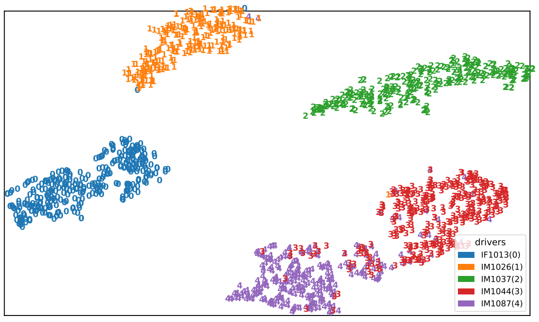
(E)



(F)



(G)



(H)

FIGURE 8.7: Visualization of embeddings for 4 experiments. Figures on each row are from the same experiment. The figures on the left are PCA projections and the ones on the right t-SNE projections.

a vehicle, which can be achieved application of clustering algorithms. This can be useful for example for car rental companies, they can know if the car was driven by two people while the contract included only one eligible driver.

It is also possible that these embeddings contain information about driving style of drivers. For instance, drivers that their embedding is close to each other probably have similar driving styles, however more work needs to be done to validate this hypothesis. If true, the *deep driver* model can be used by insurance companies to detect risky drivers.

8.5 Discussion

The *deep driver* model reduces the amount of data required for training and testing. Using GMM we could achieve accuracy of over 90% only with five minutes of training and about two minutes of data for prediction, using *deep driver* we reduce this to 10 seconds of training and prediction data. Since here we speak of the average accuracy, certainly there are cases that 10 seconds training data is sampled from a part of trip with no pedal interactions, therefore the model will be ineffective. However any driving sample which includes reasonable amount of pedal or steering wheel interaction should suffice.

The *deep driver* as presented in this chapter is built upon the work presented in the previous chapters and the hand engineered feature-set that resulted from them, this feature-set gives the *embedding net* a head start in discriminating the drivers, however it does not mean that it only works with this particular set of features. An advantage o DL approaches is that they do not need pre-defined features. DNNs, especially convolutional networks, have the capability to learn features that are equally or even more effective than hand engineered features. Therefore, in principle, instead of the current feature-set we could just use the raw sensor data, the only downside would be that perhaps more convolutional layers and more training data would be needed to achieve a similar performance.

The *deep driver* model is agnostic to the *embeddings network* used in the architecture. One can swap out the proposed residual network with any DNN architecture that can take a *block* of driving data in its input. More comprehensive architecture search can be considered as a future work. An interesting idea worth exploring is the treatment of a *block* as an image, and the use of standard image classification architectures ([25–27]) as the *embedding network.* There are two advantages in using well-known architectures, *a*) months if not years is spend crafting and studying such architectures *b*) we can use *pre-trained* models to greatly improve the accuracy and reduce the training time [153–155].

In an image classifier the first convolutional layers act as simple feature detectors, such finding edges or corners. In the deeper layers, feature-maps detect more complex patterns such as grid-like patterns or textures. Therefore, we can take a model that is pre-trained on the ImageNet ([156]) to classify object categories. Then we remove the last few layers, and replace them with more suitable layers, and plug it into *deep driver* model as the *embedding network*. Then we start training the last layers of the network as we did before; this is called *fine-tuning*. In this approach early layers are either frozen, meaning we do not update their weights or use very small learning rates. *Data augmentation* can also be used to further improve the model in case there is not enough data available for training.

We also identify some potential for further investigations of the semantic meanings of the embeddings resulting from *deep driver* model. Perhaps these embeddings can be used to construct a driver profile which encompasses driving style as well as driver identification properties.

8.6 Summary

In this chapter, we introduced the *deep driver* model, based on the Triplet network architecture. We introduced a residual DNN that is used at the part of the *deep driver* model. The *deep driver* encodes a *block* of driving data into d-dimensional *embedding* vectors. These embeddings then can be used for driver ID, driver verification and other applications. We evaluate the performance of this method for driver ID task, best results are achieved with *model-size* and *prediction-window* of 15 *blocks* which is equivalent to 13.25 s with accuracy of $0.99, 0.96$, and 0.93 for 5, 10, and 15 drivers respectively. In case of driver verification we obtain EER of $0.083, 0.101$, and 0.107 for $5, 10$, and 15 drivers (using 17 seconds of training data and 13.25 seconds for prediction). For both tasks better accuracy can be achieved as more data becomes available. We showed that *deep driver* model can achieve the same accuracy as GMM with one-tenth (1/10) of data (30 seconds as opposed to 5 minutes).

Thus far, our assumption was that we have access to vehicle operation data, such as PGP and SWA. However what if such data is not available? Is it still possible to perform driver ID in absence of CAN-data? In the next chapter, we consider the case that CAN-data is not available and provide a driver ID solution that only uses GPS data from smartphone.

Live as if you were to die tomorrow. Learn as if you were to live forever.

Mahatma Gandhi

9 Driver Identification Using GPS Data from Smartphone

MOST recent research has dealt with the data from CAN-bus with high sample-rates, sometimes attained in unrealistic scenarios such as using external sensors. These solutions require specific hardware, powerful processors, and high data-rate, which limits their deployability. There is then a real need for driver ID methods that are not hardware-dependent, and that can guarantee high accuracy even with low-rate data, which minimizes communication and storage costs. In this context, we consider in this chapter the scenario of having location-data only of vehicles, and we investigate whether this low-sample rate location data is sufficient for identifying drivers or not. Although we use an external web service to extract contextual metrics from the locations, our method stands as an end-to-end driver ID solution. In particular, we present an efficient way of encoding location coordinates and road-network links using embeddings to be used in DNN models. We evaluate the proposed method using a large set of driving data covering thousands of different drivers being active for up to more than one year in the Greater Region of Luxembourg.

The remainder of this chapter is organized as follows. Section 9.1 introduces the dataset and the related terminology. In Section 9.2, we introduce the details of the proposed methodology. Then in Section 9.3, we describe the experimental setup and the baseline methods we used for comparison. Then in Section 9.4, we present the results. In Section 9.5, we discuss the implications of this approach. Finally, in Section 9.6, we

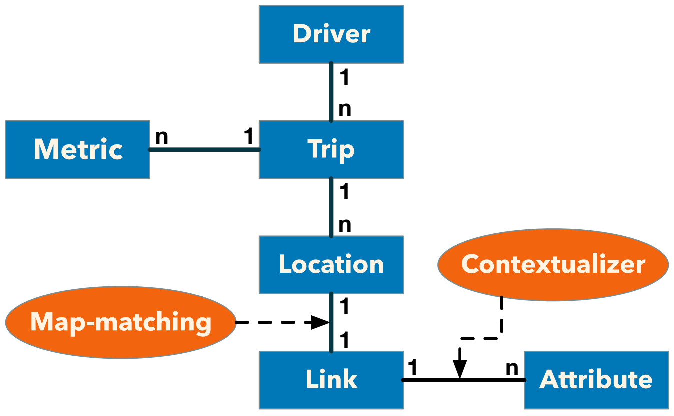


FIGURE 9.1: The relationship between different entities in the dataset. Each driver makes multiple *trips*, each *trip* is composed of multiple *location* data-points. A *map-matching* service maps a *location* data-point to one *link*, then a *contextualizer* service assigns multiple attributes to each *link*. For each *trip* we extract multiple metrics which take into account every *location*, their corresponding *links* and their *attributes*.

summarize.

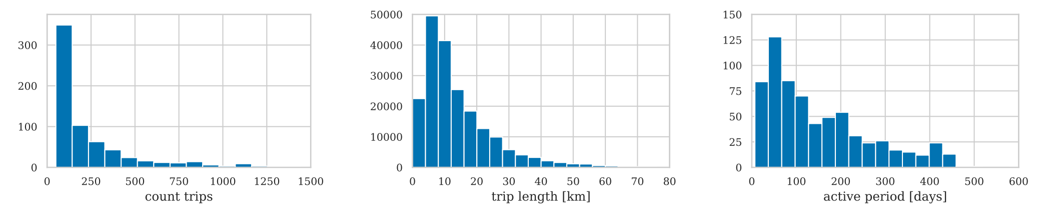
9.1 Dataset

In this work, we use a dataset kindly provided by Motion-S SA[1]. During a data collection campaign, more than ten thousand participants downloaded a gamified smartphone application to provide driving behavior tips, in exchange of receiving feedback on their driving behavior (riskiness factors) and a chance to win a prize. The participants have been collecting driving data in the background by detecting car start and stop using Bluetooth connection events with their car infotainment system. We refer to such a recording session as a *trip*. Conceptually a trip could be a commute to the work-place or back, visiting the doctor's office, or going to the mall for shopping. Additionally, there are times that the participant either disables or chooses not to record specific trips, for any possible reason.

During a trip, the app records timestamped locations updates every second. At the end of each trip or whenever the Internet connection is available, the data gets uploaded to a remote server. Location update, is comprised of latitude, longitude, altitude, speed, bearing, and estimated accuracy. The dataset used for this study is free of all personally identifiable and quasi-identifiers, we only use a pseudo-identifier to keep track of trips that belong to a *lambda* individual.

Each trip is a sequence of timestamped location data. After collection, location data are contextualized using Here.com[2] maps layer for map-matching and data augmentation. Map-matching is a process that associates a latitude, longitude coordinate to a

[1]https://www.motion-s.com
[2]Here.com `https://api.here.com`



(a) (b) (c)

Figure 9.2: *(a)* Histogram of number of trips per driver. *(b)* Histogram of trip lengths in km. There were few trips longer than 80km that are excluded. *(c)* Histogram of active days, which is the interval between the first and the last trip made by each driver. Some outliers with active period longer than 600 days are excluded.

segment of the road network, which we call a *link* and is represented by a unique integer identifier. The data augmentation process can provide a broad spectrum of attributes that characterize a particular link, e.g., slope, speed limit, administrative region, country, road roughness, points of interest, traffic signals, or whether it is a tunnel or bridge. After obtaining link attributes we compute additional features (*metrics*), e.g., speeding ratio, acceleration, and steering. An overview of these features is listed in Table 9.2.

Table 9.1: Dataset statistics

Statistic	pre-processing before	after
Number of drivers	1385	678
Average trips per driver	261.51	297.06
Total trips	362198	201410

9.1.1 Preprocessing

To ensure quality, we filter the dataset according to the following heuristics. Short trips are often faulty and do not contain enough information to be useful; therefore, the trips shorter than five minutes are removed. Moreover, we calculate the trip-length as the difference between trip-start and trip-end timestamps. There are times that although the length is over five minutes, the trace is faulty and contains few location data points, to filter such instances, we also drop trips with 100 or fewer data points. Trips having constant speed is another pattern observed among faulty trips, so we decide to remove them from the dataset. We filter out the cross-border or trips taking place outside Luxembourg territory. Lastly, we exclude drivers that have fewer than 50 trips in their records. Table 9.1 shows a summary of dataset statistics before and after preprocessing. There are 1385 drivers in our dataset, each with 13 to 5066 trips, and a total of 362198 trips. After the preprocessing steps, the number of distinct drivers reduced to 678 and a total of 201410 trips.

Table 9.2: Metrics, and dimension of their corresponding feature vectors.

	Metric	Cont.	Cat.	Seq.	Description
Temporal	Day of week	-	1 (7)	-	Day of week
	Hour	-	1 (24)	-	Hour
	Minute	-	1 (4)	-	Minutes binned into 4 quarters
	Temporal exposure	3	-	-	Slight, Serious, Fatal
	Peak-hour	-	1 (2)	-	if trip took place at peak hour.
	Total time (s)	1	-	-	duration of the trip
	daytimeproportion	1	-	-	daytime Proportion of trip
Behavioral	RPA	1	-	-	Relative positive acceleration
	Steering	1	-	-	high, Low
	Brake & Acceleration	4	-	-	high, low
	Acc./decelleration g-force	6	-	-	max, min and avg. of acc./decelleration g-force
	Count illegal actions	3	-	-	turn, u-turn, direction
	Average overspeed	3	-	-	-, curves, traffic
Spatial	Lat./Lon. of trip start/end	2×2	2×2(∼4k)	-	Latitude and longitude of start and end of trip
	Link IDs	-	-	varies	sequence of links traversed during a trip
	Bearing	1	-	-	
	Distance	2	-	-	Haversine & Manhattan distance between orig. & dest.
	Distance (km)	4	-	-	distance travelled in motorway, rural, urban areas
	Count of road features	4	-	-	Bridges, Tunnels, Urban Areas, signals, POIs
	Proportion (distance)	3	-	-	prop. of distance traveled in motorway, urban & rural
	Proportion	5	-	-	FC1, FC2, FC3, FC4, FC5
	Proportion	6	-	-	road-type (e.g. motorway, highway)
	Border crossing	1	-	-	Number of times driver crosses the border
	Number of traffic signs	6	-	-	Priority, yield, stop, lane-merge, pedestrian, overtaking
	Count of road object	3	-	-	Bridges, roundabouts, tunnels
	Count of location data points	2	-	-	Total count, valid count
	Speed in traffic	1	-	-	Driving speed in relation to traffic
	Time in traffic	1	-	-	Time spent in congested traffic

9.1.2 Features

The data set used in this chapter contains various variables describing many aspects of each trip. For that reason, in order to improve their understanding, we propose to categorize the features from two perspectives: *semantic* and *technical.* The semantic perspective is related to the nature of the metric, while the technical perspective is related to the expected type of output for that particular feature.

9.1.2.1 Semantic Categorization of Features

The act of driving implies that an individual is driving in a specific place, at a certain point of time, and behaving in a particular manner. We semantically categorize the features as follows:

Temporal: Features, derived from the date and time of the collection locations of the trip.

Spatial: Features, merely derived from specific location coordinates and the route taken by the driver.

Behavioral: Every other feature. The features that, to some extent, are a function of the driver, such as the steering and acceleration patterns.

Note that this categorization is somewhat arbitrary. One could argue that the hour of a trip should be considered behavioral because it is the driver's choice. Since our focus is driver ID and its applications, some use-cases cannot depend on every feature category. For example, in the case of long haul truck drivers, if the truck is driven by an illegitimate driver, since the destination and likely the route does not change, we cannot rely on *spatial* or even *temporal* features. Therefore *behavioral features* are the only possibility to discriminate between the two drivers. The semantic categorization of the features is indicated on the left side of Table 9.2.

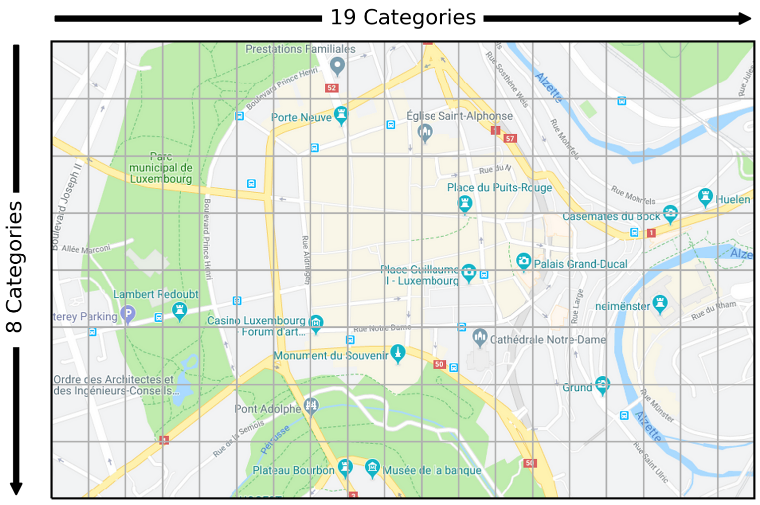


FIGURE 9.3: A demonstration of *location cross-features.* A grid is laid over the Luxembourg city's centre district. Each grid-cell can be thought of as a 2-d bucket that is assigned an integer identifier. Then any location data-point anywhere in a grid-cell is mapped to the grid-cell's identifier.

9.1.2.2 Technical Categorization of Features

The technical categorization refers to the expected output type, because every data type requires an appropriate model. There are three types of features that we are concerned with: *a*) continuous *b*) categorical *c*) sequential.

Continuous features As presented earlier, for every trip, a number of metrics is computed. We can represent real valued feature as $\boldsymbol{X} \in \mathbb{R}^{M \times N}$, where M is number of trips and N is number of features. The continuous features are scaled as below:

$$\mathbf{X}^n = \frac{\mathbf{X}^n - mean(\mathbf{X}^n)}{std(\mathbf{X}^n)} \tag{9.1}$$

$mean(\boldsymbol{X}^n)$ is the mean of the n^{th} dimension of the $\mathbf{X}$ across all trips of all drivers. $std(\boldsymbol{X}^n)$ is the standard deviation of the n^{th} dimension of the $\mathbf{X}$ across all trips of all drivers. Number of continuous features per metric is indicated in *continuous* column of Table 9.2.

Categorical these features, even though often represented as an integer type, their numerical value carries little or no meaning. For example we treat *hour* as a categorical feature, because the hours 23 and 0 even though they are farthest apart among hours of day, they both represent midnight hours. Other examples are *day of week* consisting of 7 categories or *peak hour* having only two categories.

In particular, we propose to represent *location coordinates* of origin and destination of a trip as a categorical feature. A location coordinate consists of continuous values for longitude and latitude, $\mathbf{X} = (lon, lat)$, where $X \in \mathbb{R}^2$. However, we can also treat them as a cross-feature, by binning the latitude and longitude in a grid on the map

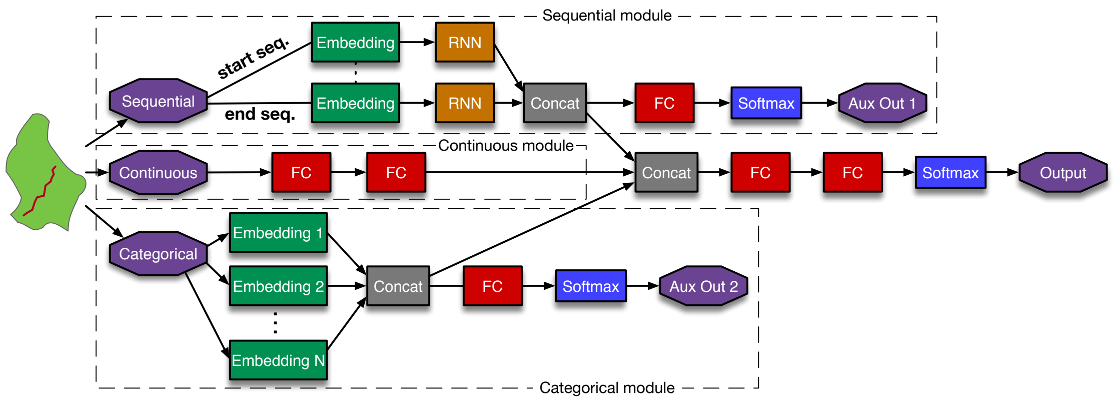


FIGURE 9.4: Architecture diagram

(see Figure 9.3). In this work we bin the coordinates to 0.003°. To do so, we assign a unique number to each cell in the grid $n \in \mathbb{N}$. Trip origin an destination coordinates are then mapped to the correspondent cell identifier $X \mapsto \mathbb{N}$, therefore every coordinate is represented by an integer. The number of categorical features are indicated in *categorical* column of Table 9.2, the number of categories are indicated between parentheses.

Sequential The only sequential feature we consider is the sequence of *links* taken by the driver. As described earlier in this Section each location is mapped to a *link*. In other words this is a sequence of categorical features. We drop the repeated consecutive *links* but still the length of the sequence varies and is trip dependent.

9.2 Methodology

The goal of driver ID is to associate a driving trip or its representation $\mathbf{x}$ to its actual driver $\mathcal{C}_k$ from a known set of K drivers. We propose to model the driver ID task as a supervised classification problem. Therefore we assume information on all possible targets, which is also known as *closed-world* scenario, assuming that all examples are from classes in the training data set. We choose a DL approach to tackle the problem. The motivation behind this decision is the ability of simultaneous handling of multiple data types. Moreover due to its modular design, more inputs of any kind can be easily added to the model.

The main challenge of this scenario is to handle sparse and high dimensional categorical data. We have two instances of such data, *links* and location cross-features. For instance, only in Luxembourg (where main part of our data is coming from) there are about 60 thousand *links*. Categorical data is generally encoded using one-hot scheme, which for a feature with n values creates a n-dimensional vector, when having features with categories in orders of thousands, one-hot encoding is not effective. To tackle the issue of high dimensionality we use a similar technique as *word embedding* [21]. Word embedding is a technique used in NLP that maps vocabulary to vectors of real numbers.

Table 9.3: Summary of model parameters for an example experiment with 50 drivers. Auxiliary outputs are omitted.

Module	Layer	Kernel, Bias	# params
Sequential	recurrent_gru_orig	(3,128),(128,)	98688
Sequential	recurrent_gru_dest	(3,128),(128,)	98688
Sequential	embed_link	(29166, 128)	3733248
Continuous	cont_dense_1	(72, 256),(256,)	18688
Continuous	cont_dense_2	(256, 128),(128,)	32896
Categorical	embed_location	(8201, 96)	787296
Categorical	embed_minute	(4, 6)	24
Categorical	embed_hour	(24, 12)	288
Categorical	embed_dayofweek	(7, 8)	56
out_branch	out_dense_1	(602, 512),(512,)	308736
out_branch	out_dense_2	(512, 128),(128,)	65664
out_branch	out_dense_3	(128, 50),(50,)	6450

We apply this technique to every categorical feature. As an example, we consider *links* to describe the method more in detail.

We can represent a trip as a sequence of links traversed, $U = \{\mathbf{u}_1, \ldots, \mathbf{u}_L\}$, where $\mathbf{u}_l$ is a one-hot vector in space Δ^D representing *link*, D is total number of links in our covered region and L is the number of links in the trip. Since D is large, learning in Δ^D space is computationally expensive. Therefore we map links into an embedding space. We call this link embedding, which is semantically similar to word embedding, $\Delta^D \mapsto \mathbb{R}^P$, where P is the dimension of embedding space that can be between 32 to 128 depending on the hyper-parameters. After applying the map $f_{link\_emb} : U \mapsto V$, becomes $V = \{\boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_L\}$, where $\boldsymbol{\nu}_l \in \mathbb{R}^P$. In practice this mapping is either pre-trained (like word vectors used in NLP) or learned jointly with the model, we chose the latter because due to relatively small amount of training data, learning an embedding jointly with model allows us to learn an embedding specialized to discriminate between drivers. On the other hand we hypothesize learning a pre-trained embedding on a large corpus (millions of trips) that is then fine-tuned for target drivers should improve the generalization and even reduce the amount of data needed for training, due to lack of sufficient data we leave this for future work [157].

We can use a similar mapping for any other categorical data, with the difference that P should be chosen proportional to the number of categories of the feature.

9.2.1 Neural Network Architecture

We propose a DNN architecture that consists of three modules, each with a different feature vector, *continuous*, *categorical* and *sequential.* Figure 9.4 illustrates the proposed architecture.

Sequential Module

The upper part of Figure 9.4 models the *link* sequence, which is the only sequential feature we consider in this work. To avoid having to handle long sequences of variable length, we separately model the beginning and end of a link sequence. We introduce a hyperparameter ρ that determines the length of the origin and destination sub-sequences. Denoting L as length of link sequence, if $L < 2\rho$ these sub-sequences may overlap and if $L < \rho$ the rest of the sub-sequence is filled with zeros. Each sub-sequence is passed through an embedding layer, these layers share their weights. The embedding vectors are initialized randomly and then trained to minimize the loss function. We feed the embeddings to the recurrent neural network (RNN). We experiment with LSTM and gated recurrent unit (GRU) cells [158, 159]. Then the output from two RNN units is concatenated and merged back to the main network. We also branch out an auxiliary output (*aux_ out1*), which consists of a fully-connected layer with Softmax activation function.

Categorical Module

For each categorical variable we create an embedding, the dimensions of this embedding is proportional to the number of categories in the variable. It is calculated using the Equation 9.2 for each categorical feature:

$$embed\_dim = \min((\lceil \log_2(vocab\_size) \rceil + 1) * \kappa, 50) \tag{9.2}$$

where $vocab\_size$ is the number of categories and κ is a hyperparameter. The resulting embeddings are then concatenated and passed through a fully-connected layer with ReLU non-linearity [150] and merge back to the main branch. Similar to *sequential branch* to facilitate learning, we also branch out an auxiliary output using a fully-connected layer and a softmax.

Continuous Module

The continuous branch is the main branch. All the continuous features are fed into two fully-connected layers and then the other two branches are merged (concatenated) in it. Then they are followed with two other fully-connected layers, a softmax activation. We concatenate outputs from these modules and feed them into two dense layers with ReLU activation functions, followed by a drop-out layer. Softmax function is applied to the output of the last drop-out layer.

Since we jointly train all modules to facilitate learning process we branch out auxiliary outputs from sequential and categorical modules, indicated as *aux_ out_ 1* and

2 in Figure 9.4. Our goal is to minimize the error:

$$\min_{f\in\mathcal{F}} \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}\left(y_n, f(X_n)\right) \tag{9.3}$$

where y is the target (correct driver) and $\mathbf{x}$ is the input feature vector and $\mathcal{L}$ measures the loss. Using the categorical cross-entropy and accommodating for the auxiliary outputs we get:

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k\in outs}\omega_k(\log p_k[y_i \in C_{y_i}]) \tag{9.4}$$

where J is the cost, to minimize. ω is the output weight and $outs$ consists of all outputs $\{main\_out, aux\_out1, aux\_out2\}$, we assign the weight 1 to the main output and 0.2 to the auxiliary outputs.

9.3 Experimental Setup

We evaluate the performance of the proposed model with real-life applications in mind. We consider scenarios with $K \in \{5, 10, 20, 50, 100\}$ drivers. Our data set contains 678 drivers, however each driver has different number of trips and their trip composition, driving style varies. To cover a representative sample of drivers we repeat experiments for 30 random subsets of drivers, each time with another random (no replacement) sample of drivers. This will also account for the case that drivers in a certain sample may be easy or difficult to discriminate.

Moreover in order to tackle the class imbalance in our data set, for every sampled driver, regardless of their number of trips, we randomly sample 200 trips (with replacement). This will lead to up or down sampling depending on how many trips a driver has in the data set. We split the 200 trips into training and validation sets in a 90%-10% ratio. Since the examples sampled for each experiment are balanced we use the accuracy as the evaluation metric:

$$acc = \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}(y_i = \hat{y}_i) \tag{9.5}$$

where $\mathbb{1}$ is the indicator function, $\hat{y}_i$ and y_i are respectively predictions and true driver for the i^{th} trip. Since for most experiments accuracy is close to 1 to improve readability we use the error-rate $(1 - acc)$. For each experiment we report the average and standard deviation of the error-rate across repetitions.

The neural network model is implemented using the functional API of Keras [160] with Tensorflow backend [161]. For training we used, batch size of 64 per GPU and optimized the model with Adam optimizer [146] for 50 epochs. The model often reaches

TABLE 9.4: Hyper parameters of the model

Module	hyper-parameter	candidate values
Sequential	link embeddings	{32, 48, 64, 96, **128**}
	recurrent unit	{**GRU**, LSTM}
	recurrent hidden units	{64, 96, **128**}
	recurrent dropout rate	{0.0, 0.3, **0.5**}
	ρ (sequence length)	{10, **15**, 20, 30}
	fully connected	{**[256, 128]**, [128, 64], [256], [128]}
Continuous	fully connected	{**[512, 256]**, [256, 128], [128, 128], [128, 64]}
	dropout rate	{0.1, 0.3, **0.5**}
Categorical	location embedding dims.	{32, 48, 64, **96**, 128}
	κ (embedding coeff.)	{1, **2**, 3}

the lowest error-rate in 20 epochs. We stored the model parameters at each epoch, and later picked the parameters with the best score on the validation set.

9.3.1 Hyperparameter Search

DL architectures have large number of hyperparameters and the search space to find the optimal architecture is computationally expensive to cover. To find hyperparameters that perform well we break the network down to its modules. For each module we perform a separate hyper-parameter search then we use the optimal parameters for the full module. Clearly with this approach we will not obtain the optimal parameters, but it should be give us a close approximate. The full set of hyperparameters we experimented with is given in Table 9.4 and the best performing parameters are indicated in boldface. We performed the hyperparameter search only over one set of randomly selected 50 drivers.

9.3.2 Baseline Algorithms

Since there are no other works that we could directly compare our approach with, we introduce two baseline algorithms. The first model is LightGBM and a HMM based model, to have a reference for the sequential module of our architecture. We also consider a meta model consisting of a combination of the two baselines as a comparable contender for our proposed DNN architecture.

9.3.2.1 LightGBM

LightGBM is a variation of gradient boosting decision tree (GBDT) and the state-of-the-art algorithm for structured data [162]. Although LightGBM can process categorical data it cannot process sequential data, therefore we do not use the sequence of *links* with this classifier. We use the following hyperparameters:

- *learning_ rate*=0.1
- *max_ depth*=30
- *min_ data_ in_ leaf*=20
- *num_ leaves*=50
- *num_ round*=500

the hyperparemeters were selected after running an extensive hyperparameter search.

9.3.2.2 HMM-based Model

To set a baseline for the recurrent module of our model, we use a sequence model based on HMM. We adapt the approach used in [163] to our driver ID application. In that work the assumption is that the trips are clustered in advance according to trip destination. Their goal was to predict the destination by identifying the cluster (hidden state). We consider the hidden state to be the driver, therefore our equivalent of the cluster is the driver. The model makes use of *link*-driver co-occurrence matrix F. $F \in \mathbb{N}^{m\times n}$ where $F_{i,j}$ is count of times driver $j \in C$ in traversed link $i \in L$.
Having constructed F, we can compute $p(l|C = c)$ and $p(C = c|l)$ for any given driver (c) and *link* (l) by employing the Equations 9.6 and 9.7.

$$\begin{aligned} p(l|C=c) &= \frac{\#\textit{trips traversing l by driver c}}{\#\textit{ trips by driver c}} \\ &= \frac{F_{l,c}}{\sum_{i=1}^{m} F_{i,c}} \end{aligned} \tag{9.6}$$

$$\begin{aligned} p(C=c|l) &= \frac{\#\textit{trips traversing l by driver c}}{\#\textit{ trips passing via l}} \\ &= \frac{F_{l,c}}{\sum_{j=1}^{n} F_{l,j}} \end{aligned} \tag{9.7}$$

Algorithm 1 shows how we perform the prediction. P_i represents likelihood of driver i being the actual driver. *Normalize* is a routine that normalizes every P_i to be a valid probability. T is total number of *links* in trip. It is possible that driver crosses a link that she never has or there could be a mistake in map-matching, to avoid getting zero probability for that presumably correct driver, the constant r is introduced alleviate this problem.

9.3.2.3 Combined Model

We also evaluate the performance of a combined model of LightGBM and the HMM model. We take a linear combination of their output probabilities and use the resulting probabilities for prediction. We introduce a hyper-parameter α to be able to adjust the

Algorithm 1 Driver prediction with drivers as hidden states

```
1: $n \leftarrow |C|$
2: $k \leftarrow 1$
3: $r \leftarrow 0.1$
4: for $i \in C$ do
5:     $P_i \leftarrow p(C = i|l_1)$
6: end for
7: while $k \leq T$ do
8:     for $i \in C$ do
9:         $P_i \leftarrow P_i \cdot p(l_{k+1}|C = i)$
10:    end for
11:    $normalize(P_i)_{i \in C}$
12:    for $i \in C$ do
13:        $P_i \leftarrow \frac{r}{n} + (1 - r)P_i$
14:    end for
15:    $k \leftarrow k + 1$
16: end while
17: return $\arg\max_{i \in C} P_i$
```

contribution of each model. Equation 9.8 demonstrates this:

$$\hat{y} = \arg\max_{c \in C}\{\alpha h_{LightGBM}(\boldsymbol{X}) + (1 - \alpha)h_{HMM}(\boldsymbol{X})\} \tag{9.8}$$

where $h_{LightGBM}$ and h_{HMM} are the LightGBM and HMM models and $\boldsymbol{X}$ is trip feature vector. Through experimentation we find the optimal value of α to be 0.7, and this is the setup we used in evaluations.

9.4 Experimental Results

This section presents the experiments performed and their results. We compare the proposed DNN architecture with baseline methods. Then we evaluate the effect of various combinations of semantic feature categories to simulate how the model would perform for various use-cases. We investigate the effectiveness of using the cross-features. And lastly we investigate the effect of amount of training data on prediction performance.

9.4.1 Comparison with Baselines

In this experiment every algorithm is fed with the same data sets and the only difference is in the features which is an artifact of inherent differences between the models. Particularly LightGBM model lacks the *link sequence* feature while HMM model only has the *link sequence* as its input feature. Figure 9.5 shows the error rates of the proposed model in compare to baselines. LightGBM and HMM models perform similarly for groups of 5 drivers, but as the number of drivers increases the HMM model significantly deteriorates

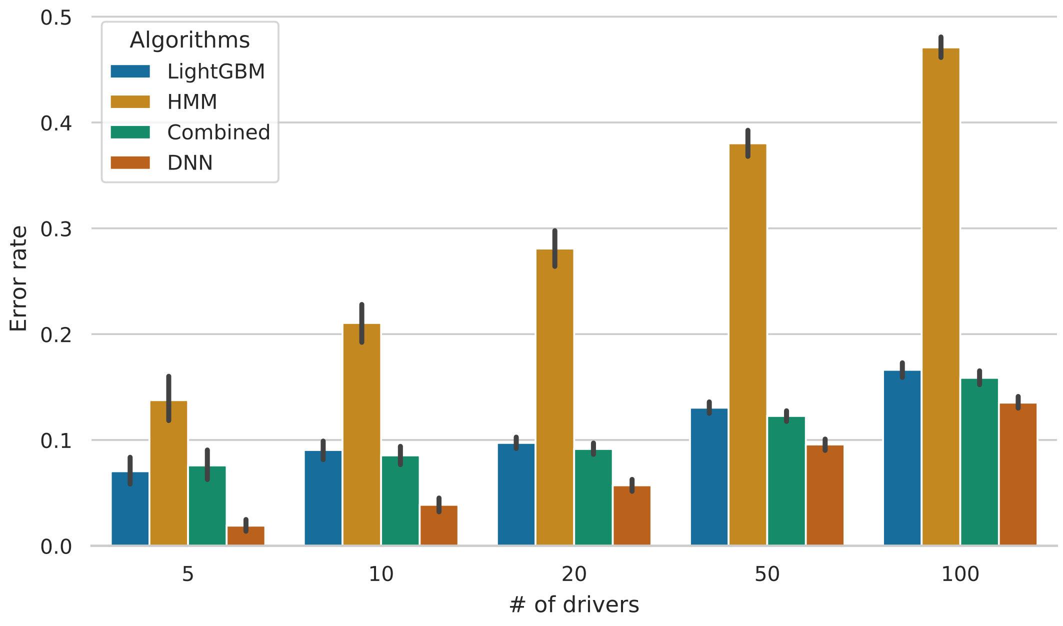


FIGURE 9.5: Error-rate comparison between baseline and DNN. *(a)* and *(b)* are different representation of the same results. HMM has the highest error-rate, LightGBM performs much better than HMM. LightGBM/HMM combined model consistently provides slightly better performance than LightGBM. It is clear that DNN approach outperforms other approaches. We also observe that for 5 drivers, there is a large performance gap between DNN and combined model, however this gap decreases as the number of drivers increase to 100.

in performance. The combined model as expected has lower error-rate for every scenario than LightGBM or HMM models. We can also see that the DNN model outperforms the baseline models in every scenario.

9.4.2 Semantic Feature Categories

In this experiment we evaluate the importance the three semantic feature categories that we introduced in Subsection 9.1.2.1. We compared all possible combinations of feature categories. To depict the model performance under various circumstances and applications. The corresponding results are presented in Figure 9.6. We can clearly see that *temporal* features alone, perform the worst with about 40% error rate for 5 drivers that increases to 79.3% for 100 drivers, this shows that merely knowing the time-of-day and day-of-week are not enough to discriminate between drivers. Second best feature category is the *behavioral*, it performs reasonably well for small number of drivers (19% error-rate for 5 drivers) but as the number of drivers increase the error rate also increases. Since behavioral features are not directly route dependent, this shows promise for applications such as theft detection or driver verification, because error-rate would even decrease further when trained with a smaller set of drivers. *Spatial* features by far are the best category, they provide error-rate of orders of magnitude lower than other

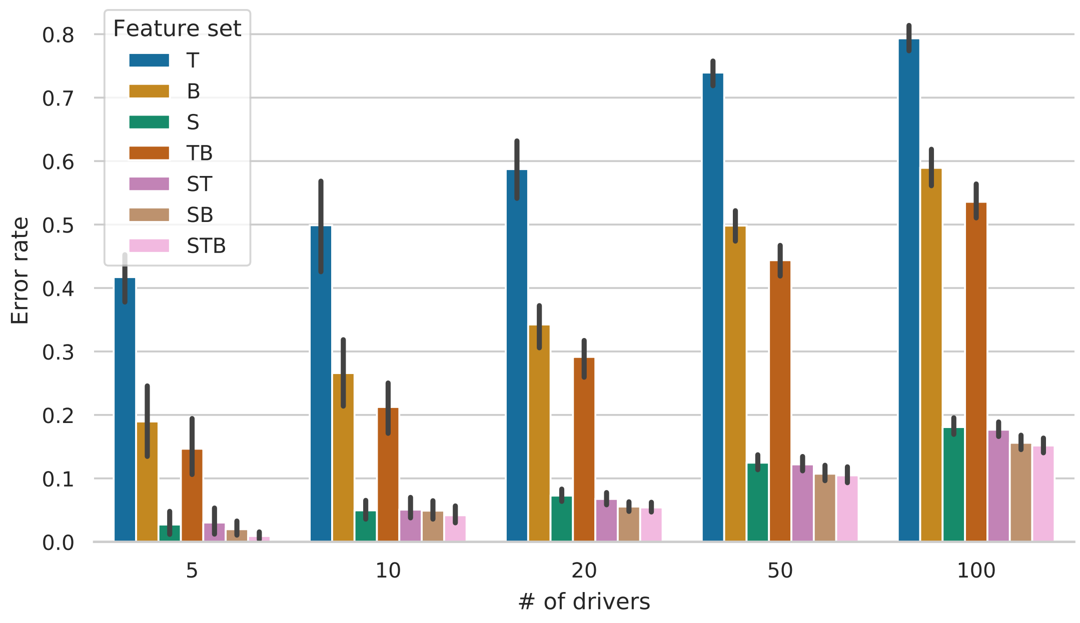


FIGURE 9.6: Error-rate per feature-set. Using all features (*STB*) results in low error-rate of 0.009 for 5 drivers and 0.152 for 100 drivers. Clearly the best single category is spatial (*S*), followed by behavioral (*B*).

feature categories, this is justified because each person only travels to a limited number of POIs and it is not difficult to learn them. It is important to note that combinations of feature categories always results in improved accuracy, this shows that all the feature categories contribute to the predictions, but their contribution is not equal. Since the contribution of temporal feature category is small we could remove them from the system without significant increase in error-rate, with nothing to lose in terms of error-rate we gain in being more privacy friendly.

Figure 9.7 shows the top 20 features obtained from LightGBM model corresponding to a randomly selected experiment. Although the order may not completely correspond to the contribution of each feature to our DNN model it will help to get an idea of the most important features. We can see latitude and longitude of origin and destination constitute the top 4 features. Then there are a number of behavioral features such as a number of *speeding* related features and features derived from acceleration patterns. The other group of features are those reflecting the composition of the route taken by the drivers, the distance and the number of *links*.

TABLE 9.5: Effectiveness of location cross-features

# drivers	Error-rate - mean (std.)		
	Full model	Excl. cross-features	Excl. coordinates
5	0.019 (0.016)	0.0211 (0.02)	0.0218 (0.023)
10	0.0387 (0.018)	0.0456 (0.02)	0.0422 (0.02)
20	0.0571 (0.016)	0.0663 (0.017)	0.0594 (0.015)
50	0.0957 (0.015)	0.115 (0.017)	0.0998 (0.017)
100	0.135 (0.016)	0.149 (0.017)	0.139 (0.015)

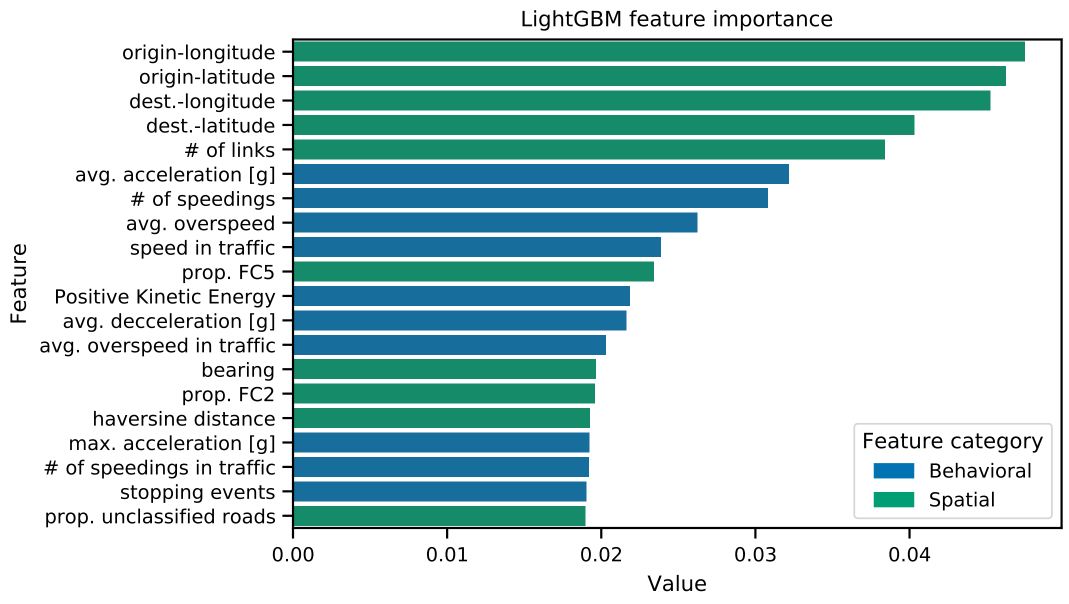


FIGURE 9.7: Top 20 most important features obtained from LightGBM. Spatial features related to origin and destination dominate the top spots, followed by even mix of spatial and behavioral features.

9.4.3 Effectiveness of Cross-features

We introduced *location cross-features* in Section 9.1.2.2 however knowing that in this work accurate origin and destination coordinates are available, this approach may seem unnecessary. Consider the case accurate coordinates are not available. This could be either due to lack of accurate data or because of privacy considerations portion of beginning and end of a trip is trimmed before upload to the server, or perhaps differential privacy mechanisms are applied to the data (perturbing the origin, destination coordinates) [164, 165]. Therefore in cases that we cannot rely on accuracy of location coordinates binning may provide a better performance because we no longer assume that coordinates are accurate measurements. To investigate effectiveness of these features we compare the model performance under three circumstances: *a*) full model including both origin, destination coordinates and cross-features *b*) full model excluding cross-features *c*) full model excluding coordinates. The second experiment will show whether addition of cross-features in presence accurate coordinates is unnecessary, and the third case will examine whether the model can perform well in absence of accurate coordinates.

Table 9.5 shows the results of the above-mentioned experiments. We can see that for any number of drivers removing the cross-features will increase the error rate. This confirms that *cross-features* contribute toward better driver ID (about 10% reduction of error-rate in average for 100 drivers). In absence of coordinates (3rd case) the model

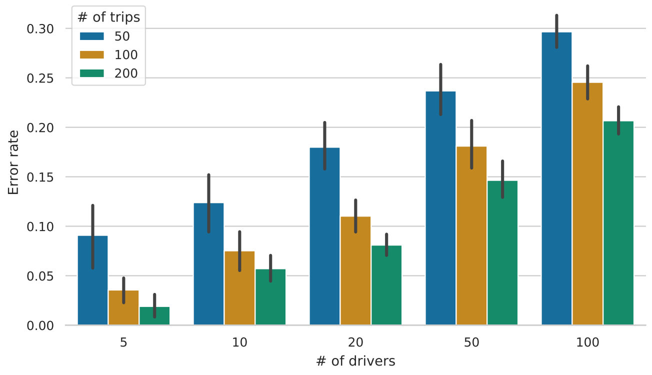


FIGURE 9.8: Error-rate for 3 experiments with 50, 100, 200 trips for training. We can clearly see that increase in amount of training data greatly reduced the error-rate. We can expect that even larger amount of training data would further decrease the error-rate.

performs slightly worse than the full model but better than the case without the *cross-features*, we believe this is due to the superior generalization capabilities of the *embeddings*. In future work, in order to further asses the robustness of the system we suggest investigating how trimming the start and end of the trips would affect the performance.

9.4.4 Sensitivity Analysis of Training-data

We investigate how the amount of training data affects the model error-rate by experimenting with $\mathcal{L} \in \{50, 100, 200\}$ trips. In these experiments at each iteration, for each driver we randomly sample $\mathcal{L}$ trips, and use this data set for training and validation. The error-rate and its standard deviation decreases inversely proportional with $\mathcal{L}$, however this effect dampens as $\mathcal{L}$ increases. There are two explanations, 1) There is only so much to learn, over that there is little information that the the model can learn from the extra training data. 2) We do not have equal number of trips from each driver, it is possible the fact that we sample the trips with replacement, negatively affects the performance for the larger values of $\mathcal{L}$, perhaps too many repeated samples.

9.5 Discussion

We motivated this work as if driver identification is a desired functionality. However one can look at it from adversarial perspective. A system such as what we discussed here can be used by an adversary as part of a surveillance system or for data de-anonymization. Assume a dataset of anonymized GPS trajectories is published online. Now if the adversary somehow gets access to identities for a subset of the dataset or even a disjoint set, as long as there are individuals that have participated in both. In such cases, our proposed DNN can be used to de-anonymize the dataset. We see the biggest potential in driver identification only using *behavioral* features. Ride-sharing services such as Uber, Lyft need to verify driver's identity to prevent fraud and ensure rider's safety. In such case, we cannot rely on *spatial* or *temporal*, unless we encounter an obvious anomaly, for example a driver that always works between 9–17h, now starts taking rides at midnight, or imagine a change in city or country. Other than such extreme examples, the focus must be on *behavioral* features. Our approach achieved 81% accuracy for 5 drivers, better results can be achieved for smaller number of drivers, or re-formulate the problem as a verification task, which is a simpler problem. For the spacial case of ride-sharing drivers, the cost of false positives is not high, a reasonable implementation can operate as follows. The DNN fails to verify the driver, then it triggers a more reliable authentication method such as facial recognition—Uber appears to already use this approach this although it is unclear what triggers it[98]—verify driver's identity, and based on that decide to escalate or resolve the issue.

Furthermore to decrease the amount of training data one can explore pre-training link embeddings on a large corpus or use techniques such as node2vec [166] that can be applied to the road network. In node2vec, instead of using real driving data, we can construct the graph representing the road network and use random walks on the graph to learn representations (embeddings) for the nodes—which we would refer to as *links* in our problem—of the graph. Lastly, it is worth exploring if it is possible to learn directly from the location data for example by application of 1-dimensional convolutional neural networks or RNNs to longitudinal and lateral acceleration estimated from speed and bearing, which is available in location data obtained from smartphones.

9.6 Summary

In this work we presented an end-to-end driver ID framework. The input to the system is location coordinates corresponding to a trip, sampled every second. This data is easy to obtain, either through a Smartphone or from the car itself. This is especially important because more and more car manufacturers are providing always on connection for their cars, and upload car operation data to their servers. Some companies such as Daimler and BMW provide APIs for third-parties to access to such data. The input is then augmented and contextualized using an external service. The augmented features have various data types, including continuous, categorical, as well as variable length sequential data. Our proposed system can accept all the above mentioned features and identify the drivers. We compare our system with three strong baselines and outperform them. Since based on use-case it may not be desirable to use every feature categories, therefore we evaluated our method using different feature sets; We conclude that *spatial* features are most effective, then the *behavioral* features and the *temporal* features are the least effective. We also investigated the effectiveness of location cross-features and showed that while using numerical value of latitude, longitude of the origin, destination and their cross-features is the most effective. When using either one of cross-features or numerical values, the former performs slightly better (up to one percentage point). We also showed that our method improves as more data is collected, which is ideal for a system that is constantly collecting more data per its usage. We obtain average error-rate of 1.9, 3.87, 5.71, 9.57, 13.5% for groups of 5, 10, 20, 50, 100 drivers which shows a great promise and enables other applications to be built on top of this system.

I refuse to answer that question on the grounds that
I don't know the answer.

Douglas Adams

10 Conclusion

IN this thesis, we examined how to enable novel applications by leveraging ML and driving data. We explored two applications, *distracted driving detection*, motivated by its potential impact on reducing road crashes and *driver ID*, which, we believe, is an enabling technology allowing a myriad of services to be built on top of it.

In Chapter 3, we introduced the distraction detection literature and identified a need for a non-intrusive distracted detection method. In Chapter 4, we presented an ML-based distracted driving detection method and showed that it is possible to detect distractions using simple driving data.

In Chapter 5, we presented the topic of driver ID and an extensive literature review. Then we proposed three driver ID methods that rely on CAN-data. First, in Chapter 6, we presented a method based on discriminative classifiers. Second, in Chapter 7, we improved upon the previous approach using a model based on the Gaussian mixture model (GMM). Third, in Chapter 8, we presented a novel approach based on deep learning. Then in Chapter 9, we presented a deep learning-based driver ID method that instead of CAN-data uses FCD obtained from smartphones, which reduces, even more the deployment complexity.

In this final chapter, we summarize our contributions, provide a final discussion, and present future research directions.

10.1 Summary of Contributions

At the beginning of this dissertation, we set out to answer the following research question:

“How could we use machine learning to reliably profile drivers?”

We decided to break this question into two more specific research questions—each dedicated to one application area, distracted driving detection and driver ID.

10.1.1 Distracted Driving Detection

Is it possible to detect distracted driving using driving behavior signals? Our first contribution, presented in Chapter 4, answers this question. We proposed a *distracted driving detection* mechanism that operates using only five signals from CAN-bus and three signals from IMU. Unlike most current solutions, this approach does not use physiological responses nor requires cameras for face/eye-tracking. The system uses a sliding window approach; it extracts statistical and cepstral features from each frame, then a discriminative classifier predicts distraction likelihoods for each frame. Then a decision function takes the per-frame predictions for the last n frames (*decision window*) and makes a final prediction. To evaluate the model, we annotated a subset of the Uyanik dataset, consisting of 13 drivers (see Section 4.2). To emulate real-life applications, we evaluated the model using a subject-independent scheme. Therefore the set of drivers used for training is disjoint from testing. The proposed solution is customizable as one can choose which classifier, *frame-size*, or *decision window* size to use. Experiments show that larger *frame-size* and *decision windows* lead to better results, but also increases the delay. *This work shows that even with less than 10 hours of coarsely annotated data and the use of vehicle sensor data, it is possible to detect distracted driving events.* This approach is limited in detecting short-term distractions. This contribution is presented in Chapter 4 and published in [53].

10.1.2 Driver Identification

How may we most effectively identify drivers based on their driving behavior? Driver ID techniques in the literature mainly use two categories of data: *a*) high sample-rate *CAN-data* *b*) low sample-rate *FCD*. We mostly focused on CAN-data and proposed three novel approaches that each incrementally addresses some of the shortcomings in state-of-the-art methods. To evaluate them, we used the Uyanik dataset consisting of 57 drivers.

State-of-the-art CAN-data methods often disregard the practicality of their solutions and use a large number of signals, or sensors that are not available on cars. Therefore, as our second contribution, in Chapter 6, we focused on finding the minimal set of signals and features that provides adequate identification performance at the same time are available on most cars. We proposed a method based on discriminative classifiers that, unlike the majority of related work that uses unrealistic sensors, only uses five signals from CAN-bus. This method uses an iterative sliding window approach and extracts features from each frame, and a classifier makes predictions per frame. Statistical features are extracted from all signals, and spectral/cepstral are extracted from the gas pedal position and steering wheel angle signals. The prediction for an ongoing driving session is provided through a decision function that takes into account the classification scores of the previous iterations. We found that spectral and cepstral features of controls operated by the driver—such as gas/brake pedals or steering wheel—are the most suitable for driver ID. The best results achieved using AdaBoost classifier with $98.86\%, 97.15\%, 93.92\%$ accuracy for $5, 15, 35$ drivers, respectively (with 20 minutes of training data and 4 minutes of prediction-data). However, this method has some limitations, namely: *a*) the need to retrain the models when enrolling new users, *b*) the need for large amounts of train and prediction data to achieve high accuracy. This contribution is published in [54].

The third contribution, presented in Chapter 7 is a driver ID algorithm based on GMM. By employing the findings from the previous approach, we decreased the number of signals from 5 to 2 and only used spectral and cepstral features from the gas pedal position and steering wheel angle. The smaller number of features (20 per signal) and the use of GMM instead of discriminative classifiers results in a lower computational complexity. Since the enrollment of new drivers only involves fitting two Gaussian mixtures (one per signal), and enrollment cost are not dependent on the number of drivers currently in the database, we claim this approach is also scalable. Moreover, the vehicle speed can be used to estimate the driving route [55], since we do not use the vehicle speed, this approach is more privacy-friendly than our previous contribution. This contribution is published in [56].

The GMM approach provides near 100% accuracy. However, its major limitation is the amount of data needed to achieve such good accuracy, both for training and at prediction time. For instance, with 5 minutes of training and 2 minutes for prediction, we obtain $90.9\%, 89.4\%$, and 86.3% accuracy for $5, 15$, and 35 drivers, respectively. To further reduce the data needs and as our fourth contribution, in Chapter 8, we presented a deep learning approach to driver ID, called the *deep driver* model. The *deep driver* model is an architecture based on *Triplet networks* [52]. It operates using the same signals and features used in the GMM approach, but the input to the algorithm is a *block* that consists of the concatenation of feature vectors from n frames. The *deep driver* encodes a *block* of driving data into a d-dimensional *embedding* vector. These embeddings can be used for driver ID, driver verification, and other applications. We evaluate the performance of this method for the driver ID task, with equivalent to 13.25 s of training and prediction data we obtained $0.99, 0.96$, and 0.93 accuracy for $5, 10$, and

15 drivers, respectively. In the case of driver verification, we obtain EER of $0.083, 0.101$, and 0.107 for $5, 10$, and 15 drivers (using 17 seconds of training data and 13.25 seconds for prediction).

Our last contribution touches on the use of FCD for driver ID. An essential step in the research since one of our goals was to work on practical and deployable solutions. In Chapter 9, we proposed a deep learning architecture composed of embedding and RNN layers that only relies on GPS data collected from smartphones. The input to the system is location coordinates corresponding to a trip, sampled every second. We contextualize each location data-points of a trip. Then we construct a set of continuous, categorical, and sequential features to represent the whole trip. We also demonstrate an efficient method to encode location-data and road-network links using embeddings. For evaluation, we used a large dataset covering 678 drivers in the Greater Region of Luxembourg provided by Motion-S[1]. This approach outperforms, LightGBM, and HMM-based baselines. We obtain an overall error-rate of 1.9, $5.71, 9.57, 13.5\%$ for groups of 5, $20, 50, 100$ drivers. This contribution is currently undergoing review as a journal article.

This work demonstrates that accurate driver identification is achievable even with low sample-rate data. The constraints imposed by the use-case and data availability negatively affect the performance; in such cases, the efficient use of the available data is crucial.

10.2 Final Discussion

The exponential increase in the amount of data generated by cars, the ubiquity of connectivity, and advances in AI allows us to explore mobility applications that never been possible. In this work, we explored two applications of ML in exploiting driving data. First, we presented a low-cost, easy to deploy, *distracted driving detection* approach. We motivated this work by the large share of low and medium-income countries of the road deaths and significant contribution of distraction to accidents. We showed that it is possible to detect long-term distractions using driving data. Such a solution can provide timely feedback to drivers and encourage them to drive safely, therefore reduce the traffic-related deaths. Moreover, since the proposed method uses minimal sensors and is trained independently of the driver, it is suitable for wide deployment.

Self-driving cars are already a reality—with some limitations—and one day they may replace conventional cars, and render our work obsolete. We predict that this transition takes decades. Moreover, countries that are the most affected by road traffic crashes are the last to transition to self-driving cars—only due to inadequate road, and telecommunication infrastructure, and, most importantly, the costs. The safety-related ADASs can already compensate for some of the errors caused by distractions or make

[1]`https://www.motion-s.com`

them irrelevant through automation; however, even in high-income countries, most people cannot afford them. Therefore, we believe, affordable safety measures such as distraction detection should be pursued and perfected. Not only distraction detection can help passengers of cars with no assistive technology. They could be used in conjunction with semi-autonomous driving features. For instance, the car detects that the driver is distracted; it assumes control in a supervisory mode, ready to take full control or take corrective measures if required.

Second, we focused on the emerging topic of driver ID. The recent uptake in the number of papers on driver ID confirms that there is a growing interest in this field. Moreover, automotive, technology and insurance companies are showing interest in this topic, either by providing datasets, funding, or resources. This includes companies such as Ford [49], Toyota [35], Samsung [42], Volkswagen and Audi [47], Yahoo!, Microsoft, Technicolor [104], General Motors [41], Insurance Box Pty [48], and Tata [110]. The possibility to identify drivers based on their driving behavior is desired by fleet managers to monitor their vehicles. Driver ID can be used for theft detection, law enforcement of heavy vehicle driving times, and allows insurance and other third-party companies to provide services that otherwise would not be possible. We explored driver identification using two kinds of data, CAN-data and FCD. Each of which have their applications, generally CAN-data approaches are more accurate and need smaller amounts of data (in terms of time duration), but they need access to the car's internal network (e.g., CAN-bus) on the other hand FCD can also be obtained from smartphones, in which case no access to the car is required, and is easy to obtain. We presented two CAN-data approaches that only need two common signals available in most mass-market cars. They are scalable and need little training data to provide accurate predictions. The GMM approach depending on the number of drivers, needs about 5–10 minutes of training-data and 2–3 minutes of prediction-data to produce an accuracy of above 95%, which is suitable for most applications. The *deep driver* model reduces the amount of data needed for training to as low as 15 seconds. This improvement comes at the cost of more computation, which is a trade-off to be considered. For instance, a car rental company can construct a model before the driver leaves their parking facilities. Embeddings obtained from the *deep driver* can also be used to provide other applications such as to infer how many drivers have driven a car. For instance, car rentals can validate if the customers respect the terms of their contracts.

It is noteworthy that both the GMM and *deep driver* approaches—presented in Chapters 7 and 8—are iterative, and the algorithm is executed every 0.25 s, we can set this interval according to available resources. Based on our experiments, larger intervals result in small accuracy degradations.

We also showed that driver ID using low sample-rate FCD is feasible. Driver ID using a smartphone (FCD) is particularly in demand because most insurance products, and gig economy companies such as Uber, and Lyft operate using a smartphone application. Therefore they all can instantly use driver ID to either prevent fraud or introduce new services such as flexible insurance policies. We obtained excellent accuracy using all features, including the *spatial features*. If we only use the *behavioral features*—features not directly related to the route—the results will be less accurate, but still practical; especially, for the gig economy use-cases that require to verify the drivers. For instance, driver ID can be used to trigger a more reliable secondary authentication method such as voice or face recognition (on the smartphone). In such a use-case, false-negatives can be tolerated.

With the decline in car ownership, more people rely on ride-sharing services such as Uber, Lyft, and Bolt for their daily needs. This leads to a further increase in demand for accurate driver ID solutions. Moreover, cars are becoming more and more connected, and connectivity and storage costs drop significantly. Since electronics are becoming cheaper, and the processors more powerful. There will be more room for sophisticated and effective driver ID solutions.

In the long-term, with the penetration of higher levels of automation, driver ID in the form presented in this work becomes more challenging. Perhaps techniques can be developed to perform driver ID only when the automation systems are disengaged. However, with level 4 and 5 of automation driver plays no role in driving, and practically there is no *human driver* to ID. Moreover, some of the applications of driver ID lose their utility if the car drives itself. Insurance coverage would only depend on the car's make, model, and perhaps software version! Furthermore, there would be no need to keep track of the working hours of truck drivers if the trucks are driving themselves. Limited forms of automation, such as ACC systems—equivalent to level 1 automation—were commercially introduced more than 25 years ago; however, they are still not standard equipment on cars. Even though the technology adoption rate has increased in recent years, the automotive industry has proved itself to be extremely slow to adopt new technologies. Considering these facts we argue that driver ID will stay relevant in the future, and there is lots of value to be gained from this technology.

10.2.1 Privacy Considerations

Car manufacturers, insurance providers, and other third-party companies already collect data from cars using telematics units, insurance dongles, and blackboxes. We are not aware of any company that provides services based on driver ID. However, given the amount of data at their disposal, they are capable of doing so. Even though we focused on positive aspects of driver ID, as we mentioned earlier in Section 5.2, driver ID poses significant privacy concerns. We acknowledge that adversaries may use it for driver fingerprinting. However, our findings enable the research community to understand and mitigate such threats. Lastly, the privacy implications of driver ID should be investigated. Driver ID has numerous benign applications, but how can we prevent the adversaries? Perhaps federated learning [167] or secure multi-party computation protocols [168] can be explored as a means to provide the same level of functionality without the model or data ever leaving the car.

10.3 Future Research Perspectives

The distracted driving mechanism we presented is only the first step. To achieve a practical solution, we need to reduce false-positives. Our work is limited in two aspects. The dataset is small (13 drivers), and the annotations are not granular. A larger, more granularly annotated dataset is needed to improve the results. With enough time and resources, it is possible to use ML/DL approaches to automate the annotation process and annotate the rest of the Uyanik dataset. Moreover, we focused on the subject-independent models because it is unrealistic to have access to annotated data from the driver. An idea worth exploring is to adapt a subject-independent model to the user's driving style. To further study the robustness of the proposed distraction detection approach, it would be worth looking into how it performs under stress or information overload. Such a study requires a rich dataset that includes contextual information such as traffic conditions, road layout, road-works. Additionally, data collection should take place in a naturalistic setup across many days to capture driving under various mental states and conditions.

We showed that driver ID approaches presented in this work perform adequately for groups of 35 or fewer drivers. However, due to dataset limitations, we cannot evaluate how accuracy would be affected by a large increase in the number of drivers. Our results show that with an increase in the number of drivers, the amount of data required to achieve high accuracy also increases. Therefore, given a fixed amount of training data, the maximum achievable accuracy is inversely proportional to the number of drivers. Although measures can be taken to adapt these solutions to perform better for a larger number of drivers, most applications only require identification among a small group of drivers. This small group of drivers is perhaps family members, a limited set of drivers eligible to drive a truck or bus over a long distance in shifts. Moreover, many use-cases such as detecting stolen vehicles require detecting the illegitimate driver, in this case, we do not need to know the thief's identity (perhaps from millions of drivers) instead we want to know whether the driver is one of the few authorized drivers or not. Therefore it is rarely needed to identify a driver from a set of hundreds or thousands of drivers.

In this dissertation, we mostly considered the closed-world driver ID scenario. However, in the real world, we often face with an open-world scenario. In this scenario, the driver may not be in our database; therefore, instead of wrongly predicting the most likely driver in our database, it is desired to predict the reject class, which is to say we do not recognize this driver. One way of tackling this issue is to use an anomaly detection approach before performing the identification. A more straightforward solution would be to make predictions only when a certain level of confidence is achieved.

In Chapter 8, we hypothesized that *deep driver* embeddings could have semantic meanings; further research is needed to validate it. Moreover, although we showed that the *deep driver* representations of each driver form a cluster, experiments should confirm if it is feasible to infer the number of drivers from the *deep driver* embeddings.

In Chapter 9, we presented a driver ID method based on FCD. The model we proposed makes predictions on a per-trip basis, which is not ideal. It is worth exploring if it is possible to learn directly from the FCD. For instance, 1-dimensional convolutional neural networks or RNNs can be used on the longitudinal and lateral acceleration estimated from the FCD. Additionally, the use of the accelerometer/gyro data from the smartphone in conjunction with GPS data can be explored to increase the accuracy and reduce the detection time. Although the smartphone-to-vehicle alignment is a challenge, it is well a researched topic [73].

There are still some fundamental questions left for future research to answer. How stable is driver behavior? For instance, to what extent driving behavior and hence driver ID accuracy is affected by circumstances such as having a bad day or being late for a meeting? Additionally, the effect of context on driving should be studied as well. Driving conditions such as road-works and weather may influence the driving style. Moreover, further research is needed to understand how the use of ADASs, such as ACC, affects the driving style and potentially influences driver ID accuracy. Another research direction to be explored is the model portability? Can we use the model obtained from driving car A to identify the same driver driving the car B? Suppose an insurance customer buys a new car. Ideally, one should be able to use the same model across vehicles. These questions are vital to the development of a robust driver ID solution.

Abbreviations

AB Adaptive boosting trees 61
ACC adaptive cruise control 18
ADAS advanced driver-assistance system 17
AI artificial intelligence ii, 3
ANFIS adaptive network-based fuzzy inference system 41
ANN artificial neural network 41
API application programming interface 2
AUC area under the curve 79

BN batch normalization 83

CAN controller area network 3
CPS cyber-physical system 2

DCT discrete Cosine transform 53
DL deep learning 3, 7
DNN deep neural network ii, 6
DSRC dedicated short-range communications 2
DWT discrete wavelet transform 42, 45

ECG electrocardiogram 18
ECU electronic control unit 2
EEG electroencephalogram 18
EER equal error rate 44
ELM extreme learning machine 41
ERPM engine revolutions per minute 11
EU European Union 1

FAR false acceptance rate 88
FC fully connected 83
FCD floating car data ii, 5
FD following distance 42
FFT fast Fourier transform 42
FRR false rejection rate 88

GB gradient boosting 61
GBDT gradient boosting decision tree 101
GDPR general data protection regulation 3
GMM Gaussian mixture model ii, 5
GNSS global navigation satellite system 4
GP percentage gas pedal 71
GPS global positioning system ii, 2
GRU gated recurrent unit 98

HMM hidden Markov model 41

ICA independent component analysis 42, 44
ID identification ii, 4
IMU inertial measurement unit ii, 4
IVDR in-vehicle data recorder 42

k-NN k-nearest neighbors 27

LDA linear discriminant analysis 41
LKA lane-keep assist 2
LSTM long short-term memory 19

MFCC Mel-frequency cepstral coefficient 42
ML machine learning ii, 3
MLP multi-layer Perceptron 41

NB naïve Bayes 41
NLP natural language processing 3, 78

OBD onboard diagnostic 35
OBU on-board unit 2
OEM original equipment manufacturer 35
OV optimal velocity 41

PCA principal component analysis 42, 44
PGP percentage gas pedal 11
POI point of interest 42

ReLU rectified linear unit 83
RF random forest 41, 42, 45
RFID radio frequency identification 34
RNN recurrent neural network 98
ROC receiver operating characteristic 21, 23

SVM support vector machine 19
SWA steering wheel angle 11
SWRS steering wheel relative speed 11

t-SNE t-distributed stochastic neighbor embedding 88

UBI usage-based insurance 37

VS vehicle speed 6

YR yaw rate 11

Bibliography

[1] World Health Organization, issuing body, and ProQuest (Firm). *Global status report on road safety 2018.* 2018. OCLC: 1123195009.

[2] Contributory factors for reported road accidents (RAS50). URL `https://www.gov.uk/government/statistical-data-sets/ras50-contributory-factors`.

[3] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos. Management of an academic hpc cluster: The ul experience. In *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*, pages 959–967, Bologna, Italy, July 2014. IEEE.

[4] Vision Zero, October 2019. URL `https://en.wikipedia.org/w/index.php?title=Vision_Zero&oldid=919053044`. Page Version ID: 919053044.

[5] EU Road Safety Policy Framework 2021-2030 – Next steps towards "Vision Zero". URL `https://ec.europa.eu/transport/road_safety/sites/roadsafety/files/move-2019-01178-01-00-en-tra-00_0.pdf`.

[6] Largest Distracted Driving Behavior Study, April 2017. URL `http://zendrive.com/blog/distracted-driving/`.

[7] Suzanne P. McEvoy, Mark R. Stevenson, Anne T. McCartt, Mark Woodward, Claire Haworth, Peter Palamara, and Rina Cercarelli. Role of mobile phones in motor vehicle crashes resulting in hospital attendance: a case-crossover study. *BMJ (Clinical research ed.)*, 331(7514):428, August 2005.

[8] Sheila G Klauer, Vicki L Neale, Thomas A Dingus, Jeremy D Sudweeks, and David J Ramsey. The prevalence of driver fatigue in an urban driving environment: Results from the 100-car naturalistic driving study. 2005.

[9] Sheila G Klauer, Feng Guo, Jeremy Sudweeks, and Thomas A Dingus. An analysis of driver inattention using a case-crossover approach on 100-car data. Technical report, 2010.

[10] Jennie Connor, Gary Whitlock, Robyn Norton, and Rod Jackson. The role of driver sleepiness in car crashes: a systematic review of epidemiological studiespublication. *Accident Analysis & Prevention*, 33(1):31 – 41, 2001.

[11] A M Williamson. Moderate sleep deprivation produces impairments in cognitive and motor performance equivalent to legally prescribed levels of alcohol intoxication. *Occupational and Environmental Medicine*, 57(10):649–655, October 2000.

[12] T. L. Bunn, S. Slavova, T. W. Struttmann, and S. R. Browning. Sleepiness/fatigue and distraction/inattention as factors for fatal versus nonfatal commercial motor vehicle driver injuries. *Accident; Analysis and Prevention*, 37(5):862–869, September 2005.

[13] European Parliament, Council of the European Union. Regulation (EC) No 561/2006 of the European Parliament and of the Council of 15 March 2006 on the harmonisation of certain social legislation relating to road transport and amending Council Regulations (EEC) No 3821/85 and (EC) No 2135/98 and repealing Council Regulation (EEC) No 3820/85 (Text with EEA relevance) - Declaration. *OJ*, L 102:1–14, April 2006.

[14] European Parliament, Council of the European Union. Regulation (EU) No 165/2014 of the European Parliament and of the Council of 4 February 2014 on tachographs in road transport, repealing Council Regulation (EEC) No 3821/85 on recording equipment in road transport and amending Regulation (EC) No 561/2006 of the European Parliament and of the Council on the harmonisation of certain social legislation relating to road transport Text with EEA relevance. *OJ*, L 60, February 2014.

[15] Ross Anderson. On the security of digital tachographs. In *European Symposium on Research in Computer Security*, pages 111–125. Springer, 1998.

[16] Michael K. Lemke, Yorghos Apostolopoulos, Adam Hege, Sevil Sönmez, and Laurie Wideman. Understanding the role of sleep quality and sleep duration in commercial driving safety. *Accident Analysis & Prevention*, 97:79–86, December 2016.

[17] Rob Cave. Lorry tachograph tampering on the rise. *BBC News*, September 2017. URL `https://www.bbc.com/news/uk-41361351`.

[18] Who's Driving You? | Promoting for-hire vehicle safety and highlighting the risks of Uber and Lyft. URL `http://www.whosdrivingyou.org/`.

[19] Alfred Ng. Uber fights off scammers every day. Here's how it learned the tricks. URL `https://www.cnet.com/news/uber-fights-off-scammers-every-day-heres-how-it-learned-the-tricks/`.

[20] G.E. Moore. Cramming More Components Onto Integrated Circuits. *Proceedings of the IEEE*, 86(1):82–85, January 1998.

[21] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*, January 2013.

[22] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv:1802.05365 [cs]*, March 2018.

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[24] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *arXiv:1901.02860 [cs, stat]*, June 2019.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6): 84–90, May 2017.

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015.

[28] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, Las Vegas, NV, USA, June 2016. IEEE.

[29] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & Deep Learning for Recommender Systems. *arXiv:1606.07792 [cs, stat]*, June 2016.

[30] Martin Wöllmer, Christoph Blaschke, Thomas Schindl, Björn Schuller, Berthold Färber, Stefan Mayer, and Benjamin Trefflich. Online Driver Distraction Detection Using Long Short-Term Memory. *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*, 12(2), 2011.

[31] Fabio Tango and Marco Botta. Real-time detection system of driver distraction using machine learning. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):894–905, 2013.

[32] Takatsugu Hirayama, Kenji Mase, and Kazuya Takeda. Detection of driver distraction based on temporal relationship between eye-gaze and peripheral vehicle behavior. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 870–875, Anchorage, AK, USA, September 2012. IEEE.

[33] Lora Yekhshatyan and John D. Lee. Changes in the Correlation Between Eye and Steering Movements Indicate Driver Distraction. *IEEE Transactions on Intelligent Transportation Systems*, 14(1):136–145, March 2013.

[34] Masahiro Miyaji, Haruki Kawanaka, and Koji Oguri. Driver's cognitive distraction detection using physiological features by the adaboost. In *2009 12th International IEEE Conference on Intelligent Transportation Systems*, pages 1–6, St. Louis, October 2009. IEEE.

[35] Chiyomi Miyajima, Yoshihiro Nishiwaki, Koji Ozawa, Toshihiro Wakita, Katsunobu Itou, Kazuya Takeda, and Fumitada Itakura. Driver modeling based on driving behavior and its evaluation in driver identification. *Proc. IEEE*, 95(2): 427–437, feb 2007.

[36] Yoshihiro Nishiwaki, Koji Ozawa, Toshihiro Wakita, Chiyomi Miyajima, Katsunobu Itou, Kazuya Takeda, Yoshihiro Nishiwaki, Koji Ozawa, Toshihiro Wakita, Chiyomi Miyajima, Katsunobu Itou, and Kazuya Takeda. Chapter 3 DRIVER

IDENTIFICATION BASED ON SPECTRAL ANALYSIS OF DRIVING. *Adv. In-Vehicle Mob. Syst. Challenges Int. Stand.*, 2007.

[37] Ines Del Campo, Raul Finker, M. Victoria Martinez, Javier Echanobe, and Faiyaz Doctor. A real-time driver identification system based on artificial neural networks and cepstral analysis. In *Proc. Int. Jt. Conf. Neural Networks*, pages 1848–1855. IEEE, jul 2014.

[38] Maria Victoria Martinez, Ines Del Campo, Javier Echanobe, and Koldo Basterretxea. Driving Behavior Signals and Machine Learning: A Personalized Driver Assistance System. In *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, volume 2015-October, pages 2933–2940, sep 2015.

[39] M V Martínez, J. Echanobe, I Campo, M.V. Martinez, J. Echanobe, and I. del Campo. Driver Identification and Impostor Detection based on Driving Behavior Signals *. *2016 IEEE 19th Int. Conf. Intell. Transp. Syst.*, pages 372–378, nov 2016.

[40] Miro Enev, Alex Takakuwa, Karl Koscher, and Tadayoshi Kohno. Automobile Driver Fingerprinting. *Proc. Priv. Enhancing Technol.*, 2016(1), jan 2016.

[41] Gorkem Kar, Shubham Jain, Marco Gruteser, Jinzhu Chen, Fan Bai, and Ramesh Govindan. PredriveID. *Proc. Second ACM/IEEE Symp. Edge Comput. - SEC '17*, 13:1–12, 2017.

[42] Byung Il Kwak, JiYoung Young Woo, Huy Kang Kim, Byung Il Kwak, JiYoung Young Woo, and Huy Kang Kim. Know your master: Driver profiling-based anti-theft method. *2016 14th Annu. Conf. Privacy, Secur. Trust. PST 2016*, pages 211–218, 2016.

[43] Saad Ezzini, Ismail Berrada, and Mounir Ghogho. Who is behind the wheel? Driver identification and fingerprinting. *Journal of Big Data*, 5(1), December 2018.

[44] Jun Zhang, ZhongCheng Wu, Fang Li, Chengjun Xie, Tingting Ren, Jie Chen, and Liu Liu. A Deep Learning Framework for Driving Behavior Identification on In-Vehicle CAN-BUS Sensor Data. *Sensors*, 19(6):1356, March 2019.

[45] Jie Chen, ZhongCheng Wu, and Jun Zhang. Driver identification based on hidden feature extraction by using adaptive nonnegativity-constrained autoencoder. *Applied Soft Computing*, 74:1–9, January 2019.

[46] Luis Moreira-Matias and Haneen Farah. On Developing a Driver Identification Methodology Using In-Vehicle Data Recorders. *IEEE Transactions on Intelligent Transportation Systems*, 18(9):2387–2396, September 2017.

[47] David Hallac, Abhijit Sharang, Rainer Stahlmann, Andreas Lamprecht, Markus Huber, Martin Roehder, Rok Sosič, and Jure Leskovec. Driver identification using automobile sensor data from a single turn. *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, pages 953–958, 2016.

[48] Jasper S. Wijnands, Jason Thompson, Gideon D.P.A. Aschwanden, and Mark Stevenson. Identifying behavioural change among drivers using Long Short-Term Memory recurrent neural networks. *Transportation Research Part F: Traffic Psychology and Behaviour*, 53:34–49, February 2018.

[49] Bo Wang, Smruti Panigrahi, Mayur Narsude, and Amit Mohanty. Driver Identification Using Vehicle Telematics Data. In *WCX 17: SAE World Congress Experience*, pages 2017–01–1372, March 2017.

[50] Daun Jeong, MinSeok Kim, KyungTaek Kim, TaeWang Kim, JiHun Jin, ChungSu Lee, and Sejoon Lim. Real-time Driver Identification using Vehicular Big Data and Deep Learning. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 123–130, Maui, HI, November 2018. IEEE.

[51] Hüseyin Abut, Hakan Erdoğan, Aytül Erçil, Baran Çürüklü, Hakkı Can Koman, Fatih Taş, Ali Özgür Argunşah, Serhan Coşar, Batu Akan, Harun Karabalkan, Emrecan Çökelek, Rahmi Fıçıcı, Volkan Sezer, Serhan Danış, Mehmet Karaca, Mehmet Abbak, Mustafa Gökhan Uzunbaş, Kayhan Eritmen, Mümin Imamoğlu, and Çağatay Karabat. Real-World Data Collection with "UYANIK". In Kazuya Takeda, Hakan Erdogan, John H. L. Hansen, and Huseyin Abut, editors, *In-Vehicle Corpus and Signal Processing for Driver Behavior*, pages 23–43. Springer US, Boston, MA, 2009.

[52] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *Similarity-Based Pattern Recognition*, pages 84–92, Cham, 2015. Springer International Publishing.

[53] Sasan Jafarnejad, German Castignani, and Thomas Engel. Non-intrusive distracted driving detection based on driving sensing data. In *Proceedings of the 4th International Conference on Vehicle Technology and Intelligent Transport Systems, VEHITS 2018, Funchal, Madeira, Portugal, March 16-18, 2018*, pages 178–186, 2018.

[54] Sasan Jafarnejad, German Castignani, and Thomas Engel. Towards a real-time driver identification mechanism based on driving sensing data. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7, Yokohama, October 2017. IEEE.

[55] Xianyi Gao, Bernhard Firner, Shridatt Sugrim, Victor Kaiser-Pendergrast, Yulong Yang, and Janne Lindqvist. Elastic pathing: your speed is enough to track you. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '14 Adjunct*, pages 975–986, Seattle, Washington, 2014. ACM Press.

[56] Sasan Jafarnejad, German Castignani, and Thomas Engel. Revisiting Gaussian Mixture Models for Driver Identification. In *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–7, Madrid, September 2018. IEEE.

[57] Chiyomi Miyajima, Takashi Kusakawa, Takanori Nishino, Norihide Kitaoka, Katsunobu Itou, and Kazuya Takeda. On-going data collection of driving behavior signals. In *In-Vehicle Corpus and Signal Processing for Driver Behavior*, pages 45–54. Springer, 2009.

[58] Wes McKinney. Data Structures for Statistical Computing in Python. pages 51–56, 2010. URL `http://conference.scipy.org/proceedings/scipy2010/mckinney.html`.

[59] William J. Horrey and Christopher D. Wickens. Examining the Impact of Cell Phone Conversations on Driving Using Meta-Analytic Techniques. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 48(1):196–205, March 2006.

[60] Jeff K. Caird, Chelsea R. Willness, Piers Steel, and Chip Scialfa. A meta-analysis of the effects of cell phones on driver performance. *Accident Analysis & Prevention*, 40(4):1282–1293, July 2008.

[61] Krsto Lipovac, Miroslav Đerić, Milan Tešić, Zoran Andrić, and Bojan MariÄĞ. Mobile phone use while driving-literary review. *Transportation Research Part F: Traffic Psychology and Behaviour*, 47:132–142, May 2017.

[62] Mélanie Née, Benjamin Contrand, Ludivine Orriols, Cédric Gil-Jardiné, Cédric Galéra, and Emmanuel Lagarde. Road safety and distraction, results from a responsibility case-control study among a sample of road users interviewed at the emergency room. *Accident Analysis & Prevention*, 122:19–24, January 2019.

[63] Michael A. Regan, Charlene Hallett, and Craig P. Gordon. Driver distraction and driver inattention: Definition, relationship and taxonomy. *Accident Analysis and Prevention*, 43(5):1771–1781, 2011.

[64] J. Engström, C. A. Monk, R. J. Hanowski, W J Horrey, J. D. Lee, D. V. McGehee, M a Regan, A Stevens, E. Traube, M. Tuukkanen, T Victor, and D. Yang. A Conceptual Framework and Taxonomy for Understanding and Categorizing Driver Inattention. 2013.

[65] John D Lee, Kristie Lee Young, and Michael Arthur Regan. *Defining driver distraction*, pages 31 – 40. CRC Press, Australia, 2009.

[66] Yanchao Dong, Zhencheng Hu, Keiichi Uchimura, and Nobuki Murayama. Driver inattention monitoring system for intelligent vehicles: A review. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):596–614, 2011.

[67] Joanne L. Harbluk, Y. Ian Noy, Patricia L. Trbovich, and Moshe Eizenman. An on-road assessment of cognitive distraction: Impacts on drivers' visual behavior and braking performance. *Accident Analysis and Prevention*, 39(2):372–379, 2007.

[68] Esa M. Rantanen and Joseph H. Goldberg. The effect of mental workload on the visual field size and shape. *Ergonomics*, 42(6):816–834, jun 1999.

[69] Chris Berka, Daniel J. Levendowski, Michelle N. Lumicao, Alan Yau, Gene Davis, Vladimir T. Zivkovic, Richard E. Olmstead, Patrice D. Tremoulet, and Patrick L. Craven. EEG Correlates of Task Engagement and Mental Workload in Vigilance, Learning, and Memory Tasks. 2007.

[70] Avinash Wesley, Philip G Hoffman Hall, Dvijesh Shastri, and Ioannis Pavlidis. A Novel Method to Monitor Driver's Distractions CHI 2010: Work-in-Progress. 2010.

[71] Huiping Zhou, Makoto Itoh, and Toshiyuki Inagaki. Influence of cognitively distracting activity on driver's eye movement during preparation of changing lanes. *Proceedings of the SICE Annual Conference*, pages 866–871, 2008.

[72] Yulan Liang and John D. Lee. Combining cognitive and visual distraction: Less than the sum of its parts. *Accident Analysis and Prevention*, 42(3):881–890, may 2010.

[73] Johan Wahlstrom, Isaac Skog, and Peter Handel. Smartphone-Based Vehicle Telematics: A Ten-Year Anniversary. *IEEE Transactions on Intelligent Transportation Systems*, 18(10):2802–2825, October 2017.

[74] Chuang-Wen You, Nicholas D. Lane, Fanglin Chen, Rui Wang, Zhenyu Chen, Thomas J. Bao, Martha Montes-de Oca, Yuting Cheng, Mu Lin, Lorenzo Torresani, and Andrew T. Campbell. CarSafe app: alerting drowsy and distracted drivers using dual cameras on smartphones. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, MobiSys '13, pages 13–26, Taipei, Taiwan, June 2013. Association for Computing Machinery.

[75] Cheng Bo, Xuesi Jian, Taeho Jung, Junze Han, Xiang-Yang Li, Xufei Mao, and Yu Wang. Detecting Driver's Smartphone Usage via Nonintrusively Sensing Driving Dynamics. *IEEE Internet of Things Journal*, 4(2):340–350, April 2017.

[76] Landu Jiang, Xinye Lin, Xue Liu, Chongguang Bi, and Guoliang Xing. SafeDrive: Detecting Distracted Driving Behaviors Using Wrist-Worn Devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(4): 144:1–144:22, January 2018.

[77] Lisheng Jin, Qingning Niu, Haijing Hou, Huacai Xian, Yali Wang, and Dongdong Shi. Driver cognitive distraction detection using driving performance measures. *Discrete Dynamics in Nature and Society*, 2012.

[78] Emre Öztürk and Engin Erzin. *Driver Status Identification from Driving Behavior Signals*, pages 31–55. Springer NY, New York, NY, 2012.

[79] Thomas G Dietterich. Machine learning for sequential data: A review. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 15–30. Springer, 2002.

[80] Lex Fridman, Daniel E. Brown, William Angell, Irman Abdić, Bryan Reimer, and Hae Young Noh. Automated synchronization of driving data using vibration and steering events. *Pattern Recogn. Lett.*, 75(C):9–15, May 2016.

[81] Gunnar Farnebäck. Two-Frame Motion Estimation Based on Polynomial Expansion. Springer, 2003.

[82] M. J. E. Savitzky, A. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Anal. Chem.*, 36, 1964.

[83] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical Review E*, 69(6):066138, June 2004.

[84] Brian C. Ross. Mutual Information between Discrete and Continuous Data Sets. *PLoS ONE*, 9(2):e87357, February 2014.

[85] Yoav Freund and Robert E Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

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

[87] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class AdaBoost. *Statistics and Its Interface*, 2(3):349–360, 2009.

[88] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[89] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[90] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[91] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[92] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel Methods in Machine Learning. *The Annals of Statistics*, 36(3):1171–1220, 2008.

[93] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, April 2011.

[94] Umut Uludag and Anil K. Jain. Attacks on biometric systems: a case study in fingerprints. In *Security, Steganography, and Watermarking of Multimedia Contents VI*, volume 5306, pages 622–633. International Society for Optics and Photonics, June 2004.

[95] J. Galbally, J. Fierrez, F. Alonso-Fernandez, and M. Martinez-Diaz. Evaluation of direct attacks to fingerprint verification systems. *Telecommunication Systems*, 47 (3-4):243–254, August 2011.

[96] Andreas Riener and Alois Ferscha. Supporting Implicit Human-to-Vehicle Interaction: Driver Identification from Sitting Postures. In *Proceedings of the First Annual International Symposium on Vehicular Computing Systems*, Dublin, Ireland, 2008. ICST.

[97] Inc. Uber Technologies. Uber's us safety report. Technical report, 2019.

[98] Engineering Safety with Uber's Real-Time ID Check, March 2017. URL `https://eng.uber.com/real-time-id-check/`.

[99] Toshihiro Wakita, Koji Ozawa, Chiyomi Miyajima, Kei Igarashi, Katunobu Itou, Kazuya Takeda, and Fumitada Itakura. Driver identification using driving behavior signals. *IEICE Trans. Inf. Syst.*, E89-D(3):1188–1194, mar 2006.

[100] Abdul Wahab, Chai Quek, Chin Keong Tan, and Kazuya Takeda. Driving profile modeling and recognition based on soft computing approach. *IEEE Trans. Neural Netw.*, 20(4):563–582, 2009.

[101] Nuttun Virojboonkiate, Adsadawut Chanakitkarnchok, Peerapon Vateekul, and Kultida Rojviboonchai. Public Transport Driver Identification System Using Histogram of Acceleration Data. *Journal of Advanced Transportation*, 2019:1–15, February 2019.

[102] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1):489 – 501, 2006.

[103] Huihuan Qian, Yongsheng Ou, Xinyu Wu, Xiaoning Meng, and Yangsheng Xu. Support Vector Machine for Behavior-Based Driver Identification System. *J. Robot.*, 2010:1–11, 2010.

[104] Cheng Zhang, Mitesh Patel, Senaka Buthpitiya, Kent Lyons, Beverly Harrison, and Gregory D Abowd. Driver Classification Based on Driving Behaviors. *Proc. 21st Int. Conf. Intell. User Interfaces - IUI '16*, pages 80–84, 2016.

[105] Minh Van Ly, Sujitha Martin, and Mohan M. Trivedi. Driver classification and driving style recognition using inertial sensors. *IEEE Intell. Veh. Symp. Proc.*, (Iv):1040–1045, 2013.

[106] Sasan Jafarnejad, German Castignani, and Thomas Engel. Towards a Real-Time Driver Identification Mechanism Based on Driving Sensing Data. *20th Int. Conf. Intell. Transp. Syst.*, (October):7, 2017.

[107] Angela Burton, Tapan Parikh, Shannon Mascarenhas, Jue Zhang, Jonathan Voris, N Sertac Artan, and Wenjia Li. Driver identification and authentication with active behavior modeling. In *Netw. Serv. Manag. (CNSM), 2016 12th Int. Conf.*, pages 388–393. IEEE, 2016.

[108] Thitaree Tanprasert, Chalermpol Saiprasert, and Suttipong Thajchayapong. Combining Unsupervised Anomaly Detection and Neural Networks for Driver Identification. *Journal of Advanced Transportation*, 2017:1–13, 2017.

[109] Szilvia Lestyán, Gergely Acs, Gergely Biczók, and Zsolt Szalay. Extracting Vehicle Sensor Signals from CAN Logs for Driver Re-identification. *arXiv:1902.08956 [cs, eess]*, February 2019.

[110] Arijit Chowdhury, Tapas Chakravarty, Avik Ghose, Tanushree Banerjee, and P. Balamuralidhar. Investigations on Driver Unique Identification from Smartphone's GPS Data Alone. *Journal of Advanced Transportation*, 2018:1–11, 2018.

[111] Fabio Martinelli, Francesco Mercaldo, Albina Orlando, Vittoria Nardone, Antonella Santone, and Arun Kumar Sangaiah. Human behavior characterization for driving style recognition in vehicle system. *Comput. Electr. Eng.*, pages 1–25, 2018.

[112] Nathanael C. Fung, Bruce Wallace, Adrian D.C. Chan, Rafik Goubran, Michelle M. Porter, Shawn Marshall, and Frank Knoefel. Driver identification using vehicle acceleration and deceleration events from naturalistic driving of older drivers. In *2017 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, pages 33–38, Rochester, MN, USA, May 2017. IEEE.

[113] Xiaoning Meng, Ka Keung Lee, and Yangsheng Xu. Human Driving Behavior Recognition Based on Hidden Markov Models. *2006 IEEE Int. Conf. Robot. Biomimetics*, pages 274–279, 2006.

[114] X Zhang, X Zhao, and J Rong. A study of individual characteristics of driving behavior based on hidden markov model. *Sensors & Transducers*, 167(3):194–202, 2014.

[115] Sara Hernández Sánchez, Rubén Fernández Pozo, and Luis Alfonso Hernández Gómez. Deep Neural Networks for Driver Identification Using Accelerometer

Signals from Smartphones. In Witold Abramowicz and Rafael Corchuelo, editors, *Business Information Systems*, volume 354, pages 206–220. Springer International Publishing, Cham, 2019.

[116] SangJo Choi, JeongHee Kim, DongGu Kwak, Pongtep Angkititrakul, and John H L Hansen. Analysis and classification of driver behavior using in-vehicle can-bus information. *Bienn. Work. DSP In-Vehicle Mob. Syst.*, (October 2015):17–19, 2007.

[117] Abdul Wahab, Tan Chin Keong, Hüseyin Abut, and Kazuya Takeda. Driver Recognition System Using FNN and Statistical Methods. *Adv. In-Vehicle Mob. Syst.*, pages 11–23, 2007.

[118] Pantaree Phumphuang, Pongpisit Wuttidittachotti, and Chalermpol Saiprasert. Driver identification using variance of the acceleration data. In *2015 International Computer Science and Engineering Conference (ICSEC)*, pages 1–6, Chiang Mai, Thailand, November 2015. IEEE.

[119] Chalermpol Saiprasert and Suttipong Thajchayapong. Remote Driver Identification Using Minimal Sensory Data. *IEEE Communications Letters*, 19(10):1706–1709, October 2015.

[120] Haneen Farah, Oren Musicant, Yaara Shimshoni, Tomer Toledo, Einat Grimberg, Haim Omer, and Tsippy Lotan. The First Year of Driving: Can an In-Vehicle Data Recorder and Parental Involvement Make it Safer? *Transportation Research Record: Journal of the Transportation Research Board*, 2327(1):26–33, January 2013.

[121] Eduardo Romera, Luis M. Bergasa, and Roberto Arroyo. Need data for driver behaviour analysis? Presenting the public UAH-DriveSet. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 387–392, Rio de Janeiro, Brazil, November 2016. IEEE.

[122] Toshihiro Wakita, Koji Ozawa, Chiyomi Miyajima, Kei Igarashi, Katunobu Itou, Kazuya Takeda, and Fumitada Itakura. Driver identification using driving behavior signals. *IEICE Transactions*, 89-D(3):1188–1194, 2006.

[123] Douglas A. Reynolds. Speaker identification and verification using Gaussian mixture speaker models. *Speech Commun.*, 17(1-2):91–108, aug 1995.

[124] Fridulv Sagberg, Selpi, Giulio Francesco Bianchi Piccinini, and Johan Engström. A review of research on driving styles and road safety. *Human factors*, 57(7): 1248–1275, 2015.

[125] A Michael Noll. Cepstrum pitch determination. *The journal of the acoustical society of America*, 41(2):293–309, 1967.

[126] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In David B. Lomet, editor, *Foundations of Data Organization and Algorithms*, pages 69–84, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[127] R. B. Blackman and J. W. Tukey. The measurement of power spectra from the point of view of communications engineering - Part I. *The Bell System Technical Journal*, 37(1):185–282, January 1958.

[128] J. Makhoul. A fast cosine transform in one and two dimensions. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):27–34, February 1980.

[129] D. Opitz and R. Maclin. Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligent Research*, 11, 1999.

[130] Gilles Louppe, Louis Wehenkel, Antonio Sutera, and Pierre Geurts. Understanding variable importances in forests of randomized trees. In *Advances in neural information processing systems*, pages 431–439, 2013.

[131] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632. ACM, 2005.

[132] D.A. Reynolds and R.C. Rose. Robust text-independent speaker identification using Gaussian mixture speaker models. *IEEE Transactions on Speech and Audio Processing*, 3(1):72–83, January 1995.

[133] Douglas A. Reynolds, Thomas F. Quatieri, and Robert B. Dunn. Speaker Verification Using Adapted Gaussian Mixture Models. *Digital Signal Processing*, 10(1-3): 19–41, January 2000.

[134] S. Furui. Speaker-independent isolated word recognition based on emphasized spectral dynamics. *ICASSP '86. IEEE Int. Conf. Acoust. Speech, Signal Process.*, 11:1991–1994, 1986.

[135] B.A. Hanson and T.H. Applebaum. Robust speaker-independent word recognition using static, dynamic and acceleration features: experiments with Lombard and noisy speech. *Int. Conf. Acoust. Speech, Signal Process.*, pages 857–860, 1990.

[136] Mingzhou Song and Hongbin Wang. Highly efficient incremental estimation of gaussian mixture models for online data stream clustering. In *Intelligent Computing: Theory and Applications III*, volume 5803, pages 174–184. International Society for Optics and Photonics, 2005.

[137] David Hallac, Suvrat Bhooshan, Michael Chen, Kacem Abida, Rok Sosic, and Jure Leskovec. Drive2vec: Multiscale State-Space Embedding of Vehicular Sensor Data. *arXiv:1806.04795 [cs, stat]*, June 2018.

[138] HaiLong Liu, Tadahiro Taniguchi, Tosiaki Takano, Yusuke Tanaka, Kazuhito Takenaka, and Takashi Bando. Visualization of driving behavior using deep sparse autoencoder. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 1427–1434, MI, USA, June 2014. IEEE.

[139] Kang Liu, Song Gao, Peiyuan Qiu, Xiliang Liu, Bo Yan, and Feng Lu. Road2vec: Measuring Traffic Interactions in Urban Road System from Massive Travel Routes. *ISPRS International Journal of Geo-Information*, 6(11):321, October 2017.

[140] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature Verification using a "Siamese" Time Delay Neural Network. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 737–744. Morgan-Kaufmann, 1994.

[141] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

[142] H. Dang and J. Fürnkranz. Driver Information Embedding with Siamese LSTM networks. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 935–940, June 2019.

[143] S. Chopra, R. Hadsell, and Y. LeCun. Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546, San Diego, CA, USA, 2005. IEEE.

[144] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, volume 2, pages 1735–1742, New York, NY, USA, 2006. IEEE.

[145] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, June 2015.

[146] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014.

[147] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[148] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. page 24.

[149] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015.

[150] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[151] T. Cover and P. Hart. Nearest Neighbor Pattern Classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, September 2006.

[152] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[153] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[154] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. *arXiv:1310.1531 [cs]*, October 2013.

[155] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *arXiv:1312.6229 [cs]*, February 2014.

[156] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[157] Yoshua Bengio. Deep Learning of Representations for Unsupervised and Transfer Learning. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, UTLW'11, pages 17–37. JMLR.org, 2011. event-place: Washington, USA.

[158] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[159] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]*, June 2014.

[160] François Chollet et al. Keras. `https://keras.io`, 2015.

[161] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[162] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017.

[163] Yassine Lassoued, Julien Monteil, Yingqi Gu, Giovanni Russo, Robert Shorten, and Martin Mevissen. A Hidden Markov Model for Route and Destination Prediction. *arXiv:1804.03504 [physics]*, March 2018.

[164] Cynthia Dwork. Differential Privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer Verlag, July 2006.

[165] Miguel E. Andrés, NicolÃąs E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Geo-indistinguishability: differential privacy for location-based systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*, pages 901–914, Berlin, Germany, 2013. ACM Press.

[166] Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, pages 855–864, San Francisco, California, USA, 2016. ACM Press.

[167] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR.

[168] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012*, volume 7417, pages 643–662. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.