h e g

Haute école de gestion
Genève

Carpoolers classification while preserving users privacy

Lucas Gülen

Master thesis for :

Master in Information Science - Data science

Supervisor : **Professor Douglas Teodoro**

August 17th 2020

Haute École de Gestion de Genève (HEG-GE)

Hes·so GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

Disclaimer

This report is submitted as part of the final examination requirements of the Haute école de Gestion de Genève, for the Master's of Science HES-SO in Information Science.

The use of any conclusions or recommendations made in or based upon this report, with no prejudice to their value, engages the responsibility neither of the author, nor the author's mentor, nor the jury members nor HEG or any of its employees.

The student has sent this document by email to the address given by his master's thesis supervisor for analysis by the URKUND plagiarism detection software, according to the detailed procedure at the following URL: https://www.urkund.com .

"I certify that I have done this work alone, without having used sources other than those cited in the bibliography."

Done in Geneva, August 17, 2020

Lucas Gülen

Acknowledgments

First of all, I would like to thank my thesis supervisor, Dr. Douglas Teodoro, for his guidance, his advice and for all the ideas for exploration related to machine learning models and data throughout my thesis. Then, I would like to thank the company Mobilidée, more specifically Michalis Giannakopoulos Rochat, for having given me the opportunity to work on one of their business problems, for having welcomed me within their team and for having always been available to answer my questions related to the different problems that this thesis aimed to answer. I would also like to thank my friends and colleagues, Flavio Barreiro Lindo and Alex Perritaz, for supporting me, reading my thesis and advising me on some technical points related to machine learning. Moreover, I would especially thank Flavio Barreiro Lindo for the Portuguese cuisine that he made me taste and that allowed me to refocus on my work. Finally, I would like to thank my girlfriend, my friends and my family for their support and encouragement throughout this work and my studies.

Abstract

Carpooling is a mode of transportation that allows users to share places in their cars to reduce greenhouse gas emissions. Many services exist to encourage users to practice carpooling, while Mobilidée proposes a novel idea with a mobile application presented to other companies who want to promote soft mobility among their employees. Currently, Mobilidée already has an application that lets users enter manually whom they carpool with hand for how long. The advantage for any user is that the more they reports their carpooling data in the application, the more they will have a chance to receive incentives from his company. Thus, to ensure security in this process as much as possible, Mobilidée needs to automate data collection and analysis of their users' smartphone sensors in order to determine how much a given user has made carpooling in a given period. Indeed, today there is not a verification process in place that prohibits a user from claiming to carpool when, in reality, he does not. Hence, this work aims to investigate machine learning algorithms in order to classify people who made carpooling together or not automatically. This work will also explore some machine learning techniques to encode user data before their analysis to ensure their privacy.

Contents

List of Tables

List of Figures

List of Equations

Nomenclature

Greek symbols

β	Weights in linear regression
β_0	Bias in linear regression
δ	Significance threshold
μ	Refers to mean
∇	Refers to the gradient of any function that follows
∂	Refers to partial derivative of any term that follows
σ	Refers to the standard deviation or to the Sigmoid function in the machine learning context
θ, φ	Generally refers to any machine learning model or function parameters

Roman symbols

$\hat{y}$	Model predictions
i	Variable used as a row index
j	Variable used as a column index
L	Refers to a general loss function used in machine learning
n	Variable used as the total number of values
P	Refers to the probability of any term that follows
t	Refers to a specific timestep in recurrent artificial neural networks
W, V, U	Matrix of weights, commonly used in machine learning models
X	Matrix of inputs, commonly used in machine learning models
x	General notation for inputs data in machine learning models. Depending on the context, it could be also one of a smartphone sensor axis.
x'	Reconstructed output from an autoencoder or from a variational autoencoder
y	True predictions in supervised machine learning models. Depending on the context, it could be also one of a smartphone sensor axis.
z	Latent representation in autoencoder and variational autoencoder machine learning models. Depending on the context, it could be also one of a smartphone sensor axis.

1 Introduction

1.1 Context

The subject of this thesis is to investigate about assessing a carpooling classification problem without disclosing personal information. Carpooling is a mode of transportation that allows users to share places in their car in order to reduce greenhouse gas emissions [1]. Many services exist to encourage users to practice carpooling, such as BlaBlaCar and Waze [2], [3]. A novel idea is proposed by Mobilidée with a mobile application [4] that is presented to other companies who want to promote soft mobility among their employees.

Currently, Mobilidée already has an application that lets users enter manually whom they carpool with and for how long. The advantage for any user is that more he reports its carpooling data in the application, more he will have a chance to receive incentives from his company. Thus, to ensure security in this process as much as possible, Mobilidée needs to automate data collection and analysis of their users' smartphone sensors to determine how much a given user has made carpooling in a given period. On one hand, it will allow for fewer user interactions with the application and will decrease cheating probabilities. Indeed, today there is not a verification process in place that prohibits a user from claiming to carpool when, in reality, he does not.

1.2 Objectives

An important Mobilidée's principle is to promote soft mobility among their employees, but among their proposed services too. So, a Mobilidée objective is to be able to encode user sensor's data directly on the mobile before sending them to the server for further analysis.

Another significant Mobilidée's principle is to protect as much as possible their customer's data privacy. In this respect, one of these thesis' objectives is to be able to encode sufficiently user's data collected to ensure the maximum privacy possible.

Moreover, the automated verification process mentioned in section 1.1 is not in place in Mobilidée. Thus, another objective in this thesis is to be able to use user data to determine whether a user made carpooling or not.

In addition, we do not have a data set at our disposal to reach the objectives cited above. Thus, as we aim to build some statistical models to achieve the two firsts objectives, we must find a way to gather a sufficient amount of data to obtain satisfying results. So this objective contains two sub-

objectives:

- Collecting and constructing a data set of lower-invasive mobile phone sensor data,
- Use mobile sensors that consumes a small amount of battery power.

1.3 State-of-the-art

Data privacy

Data protection has become a central issue in the use of personal data processing systems due to the implementation of the General Data Protection Regulation (GDPR) in Europe since 2018 [5]. To address this issue, an approach has been adopted by Malekzadeh *et al.* They used a mobile autoencoder to successfully encode sensor data from 24 users with a re-identification rate of 94% by their decoder and 7% of re-identification rate by a third party system which had no access to their trained decoder [6].

Malekzadeh *et al.* also published a scientific paper in which they used a replacement autoencoder and an anonymizing autoencoder to transform sensor data to preserve confidentiality and usefulness in disclosure. They managed to maintain a confidentiality loss of data of 7% and preserve their usefulness of 92% [7].

To preserve the confidentiality of banking data, Maina *et al.* explored a type of algorithm called variational autoencoder (VAE). They managed to anonymize their data and maintain a Jaccard index ranging from 0.9 to 1 between the original data and the synthetics generated by the VAE [8]. The Jaccard index is a measure of similarity used by subtracting it from the number 1 to give the Jaccard distance. The closer this distance is to 1, the more substantial the similarity is.

Abay *et al.* also used a VAE in their research paper. Their goal was to use confidential data to generate synthetic ones following the same distribution, thus being usable in the same purpose and without any confidentiality constraint. With their model, they were able to generate synthetic data and obtained a lower classification error rate on these data than most of the current synthetic data generation got from another state-of-the-art algorithm [9].

To anonymize patient gait data, Ngoc-Dung *et al.* proposed an approach using a convolutional neural network. The network receives two gaits as input, the patient one to be anonymized, and another called "noise gait" and outputs the anonymized resulting data. Their method allowed them to achieve a failure rate in re-identification by third parties of 98.86% [10]. Another approach was used by Ab-

drashitov and Spivak, who used genetic clustering algorithms. In particular, they used L-diversity and k-anonymization to anonymize their data [11].

Time-series classification

One of the objectives of this research project is to be able to determine, based on mobile sensor data, whether two or more people are travelling in the same vehicle. Therefore, our approach will be to classify travelers using time series data from smartphone sensors. Thus, Kuk *et al.* have adopted an approach to classify time series data such as those from the mobile sensor data. They used mobile data from the accelerometer of many users travelling in a train, but in separate wagons, to assess the users were travelling together. Through using the Euclidian distance between users, they have achieved to have 93% of correct classifications [12].

In another approach, Nguyen *et al.* used magnetometer data from several London Underground users to classify people travelling together. They obtained a rate of correct classifications of 100% by using the Euclidean distance on the user magnetometer aligned data [13]. Lester *et al.* used accelerometer data from two smartphones to determine whether they were worn by the same person and in the same body location. Using the Fast Fourier Transform algorithm to convert the discrete accelerometer data into frequency data, they used a consistency function to measure the similarity of two frequency signals and achieved 100% accuracy over an average window of eight seconds of recorded data [14].

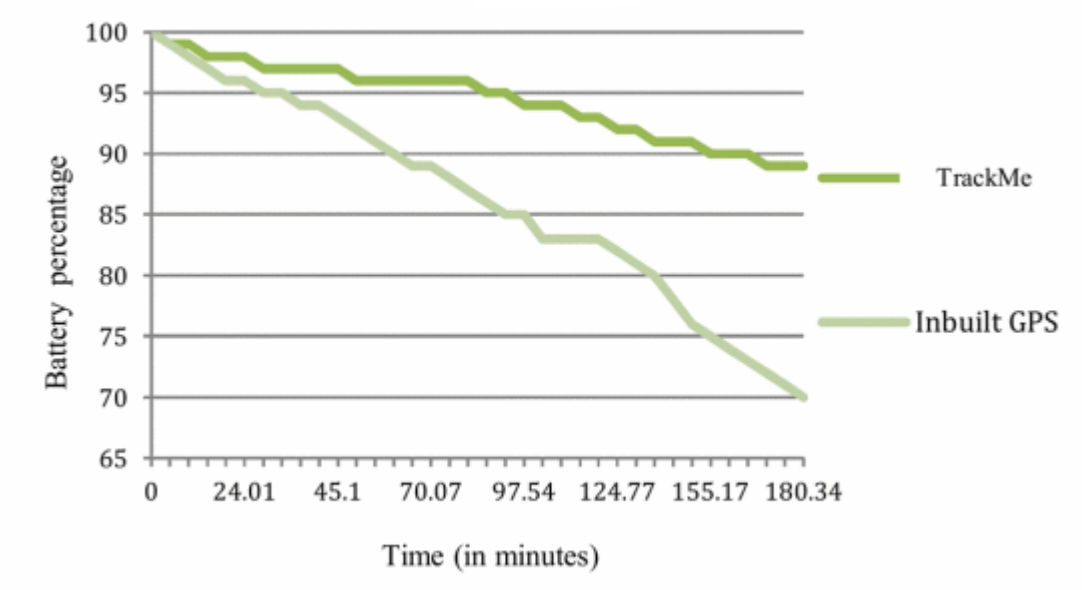
In order to recognize what activity a person was performing, based on the accelerometer, Zeng *et al.* used a convolution neural network. Their results were over 88% accurate [15].

Environmental concerns

In addition to being able to assess who has travelled together and how long they have travelled, this thesis aims to use sensors that use as least energy as possible and protect the privacy of people. As GPS is a sensor that consumes more energy than the accelerometer and reveals more confidential data about a user, such as their daily journeys, [16], it will only be used for comparison purposes in this work.

Moreover, according to Subhanjan *et al.*, GPS would consume more smartphone battery than the magnetometer and the accelerometer combined. To reach this conclusion, they carried out tests on the use of GPS alone and the use of the magnetometer and accelerometer in another case, over 180 minutes [17]. Here is the graph they published in their scientific paper, where the "TrackMe" curve represents the use of the magnetometer and the accelerometer through their mobile application :

Figure 1.1: GPS battery consumption - Magnetometer and accelerometer battery consumption



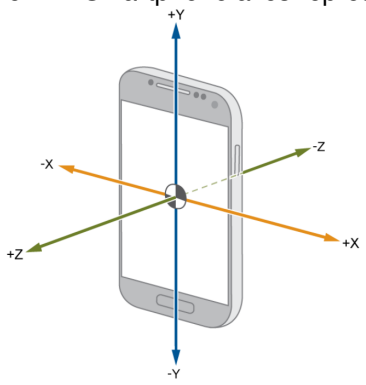
Furthermore, according to Panichpapiboon *et al.*, the accelerometer's use as a substitute for the GPS sensor to estimate the average speed of a moving vehicle is possible and more accurate [18].

2 Materials and methods

2.1 Data

For this thesis, three mobile sensor's data were chosen: the gyroscope, the accelerometer, and the magnetometer. Since they are motion sensors commonly used for user activity detection and user identification [14] [16] [13], they were chosen over other sensors such as the barometer, proximity sensor, and ambient light sensor. Each one has three axes, represented as x, y, z, and can be interpreted as the three-dimensional space where the smartphone is located. Moreover, these three axes are in a fixed position relative to the smartphone, whatever the rotation or the force applied. Here is a graphical representation of the three-dimensional space used by all sensors in this thesis:

Figure 2.1: Smartphone axes representation



[19]

The accelerometer is one of the sensors which will be used. It measures how fast the acceleration changes, including physical acceleration and gravity, at a given time in m/s^2[20] [21] in x, y and z axis and is given by the following formula:

$$\frac{distance}{time} = speed. \tag{2.1}$$

There are two common types of accelerometers on the market: capacitance and Piezoelectric accelerometer. Usually, most of the commercialized smartphones use capacitive based techniques [22]. It works by sensing changes in capacitance between microstructures next to the accelerometer. Then it is translated into voltage for correct interpretation.

The magnetometer is the second sensor, and it measures magnetic fields along three axes x, y, z

and the total magnitude of the magnetic field. Its units are micro Tesla and are written as μT. The magnetic forces applied on the mobile sensor's magnetometer are due to the magnetic effect generated by electric currents, magnetic materials near the sensor or Earth's magnetic force [23].

Gyroscope is the third mobile sensor used in this work. It senses angular rotational velocity and acceleration [24]. All measures are reported in the x, y and z fields and values are expressed in radians per second (rad/s) [20].

In this work, data from each sensor above is collected sequentially in different timestamps. It means that each data interval collected will have n frames of data where each frame has x, y and z values for each sensor. Thus, it could be interpreted as a time series where frame values are ordered by timestamp. Here is an example of mobile accelerometer's data collected in 10 frames with a capture interval of $500ms$:

Table 2.1: Mobile sensor's values example

	Accelerometer		
Frame number	x	y	z
1	0.04	4.32	9.75
2	6.91	0.55	4.18
3	1.92	-5.14	11.11
4	2.43	-1.65	9.05
5	2.73	-1.53	9.34
6	2.51	-1.70	9.31
7	2.55	-1.61	9.38
8	2.47	-1.71	9.20
9	2.44	-1.65	9.29
10	2.42	-1.56	9.39
n=10			interval=500ms

In Table 2.1, a minor variation can be observed in z axis which approximates the Earth's standard acceleration gravity value of $9.807m/s^2$ [25]. One might assume that the z axis is facing the Earth's core and so the smartphone screen too, according to Figure 2.1. This assumption could have been done on y axis if the smartphone was in a position where y was facing the Earth's core.

The collected data are organized by travel with always a pair of one driver data matrix and one passenger data matrix. Each one has 12 features and is constructed like the sample of data shown in Table 2.2 below :

Table 2.2: Passenger's sample data matrix

Accelerometer			**Gyroscope**			**Magnetometer**				
x	y	z	x	y	z	x	y	z	Magnitude	Label
0.04	4.32	9.74	-0.19	-2.02	-0.60	1.50	2.81	-28.31	28.49	0
6.91	0.55	4.18	-2.19	0.50	0.22	-19.75	8.63	-17.81	27.96	0
1.92	-5.14	11.11	1.74	-1.26	-0.05	-22.25	7.69	-14.88	27.85	0
2.43	-1.65	9.05	1.74	-1.26	-0.05	19.88	6.19	-9.56	22.91	0
2.73	-1.53	9.34	1.74	-1.26	-0.05	4.00	12.81	-8.75	16.02	0
2.51	-1.70	9.31	1.74	-1.26	-0.05	-4.63	11.69	-9.75	15.91	0
2.55	-1.61	9.38	1.74	-1.26	-0.05	-7.44	8.50	-5.94	12.76	0
2.47	-1.71	9.20	1.74	-1.26	-0.05	-6.81	8.13	-5.38	11.89	0
2.44	-1.65	9.29	1.74	-1.26	-0.05	-5.94	7.94	-5.19	11.19	0
2.42	-1.56	9.39	1.74	-1.26	-0.05	-6.25	8.25	-5.56	11.75	0

All statistics and figures in section 2.1 will use the passenger file who is sampling in Table above. Each line of the Table 2.2 above represents a capture time (frame) for the passenger's travel where each accelerometer, gyroscope, and magnetometer axes values are recorded. The last column is the label that could be "True" or "False" (1 or 0). When it is true, it means that the passenger and the driver of this trip were together at the current frame. Data used in Table 2.2 do not contain the timestamp field because of presentation concerns.

For each carpooling trip where there are more than two passengers, it is split into $n-1$ trips, where n is the number of people in the car using the application. Every person is paired with the driver in order to simplify the data structure for the classification task. In fact, instead of mapping one driver with multiple passengers, it makes all combinations between the driver and the passengers to get $n-1$ trips of exactly one driver-passenger pair.

2.1.1 Descriptive analysis

This section will discuss data distribution and try to make some preliminary assumptions. A critical thing to verify is that all classes are balanced as much as possible in our dataset. Thus, according to Batista *et al.* [26], some statistical models in machine learning tasks may compute much better results on a balanced dataset.

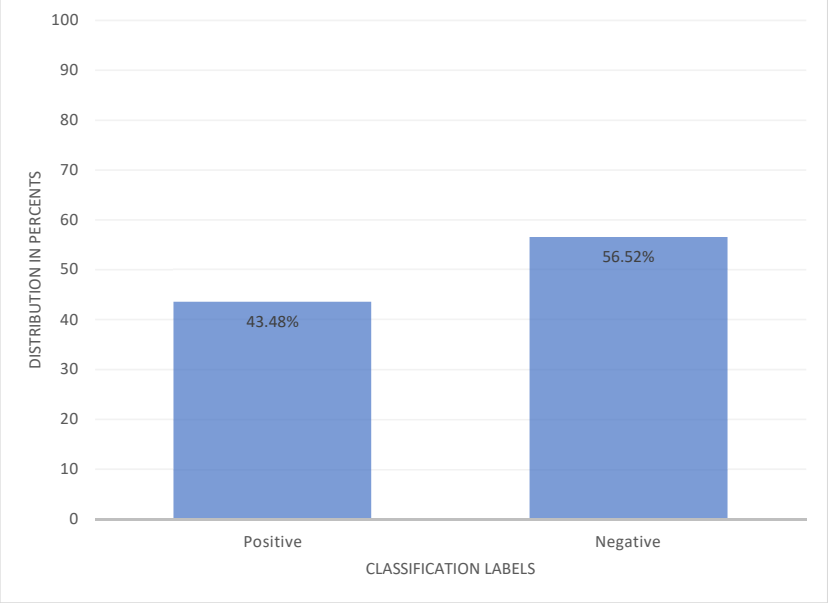
In this thesis, the classification task is a binary classification problem where a label can be either 1 (True / Positive) or 0 (False / Negative). The "Positive" value for a trip indicates that the passenger and the driver were together. Figure 2.2 below shows the proportion of each class over all trips in the data set. Therefore, there are 56.52% of negative instances where the driver and the passenger have generated data, but in two distinct vehicles. Hence, it could be assumed that the data set is balanced

in an acceptable manner where there is not a large gap between each class, given the 23 trips in total as seen in Table 2.3. The Table 2.3 also shows the minimum, maximum and the average of frames with respect to the entire dataset of trips.

Table 2.3: Dataset summary

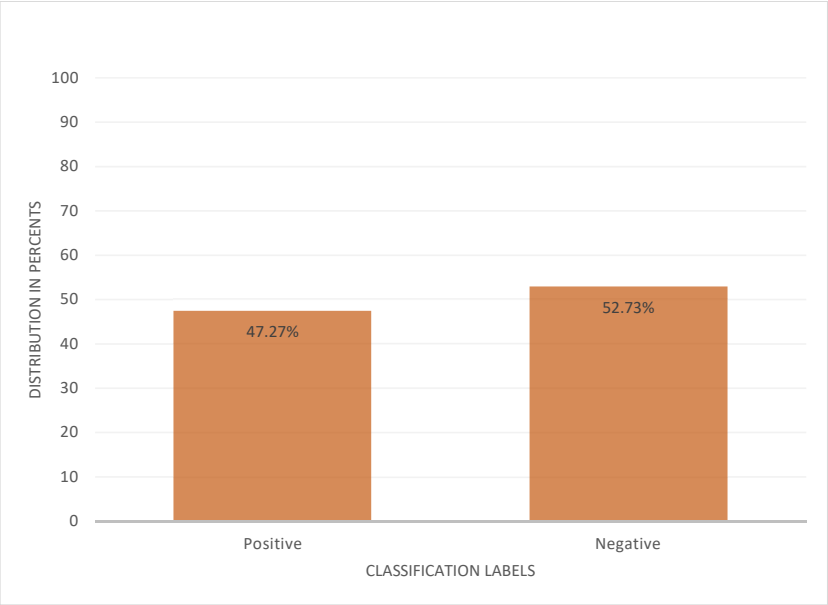
Number of trips	**Min. frames**	**Max. frames**	**Avg. frames**
23	3751	6508	5348

Figure 2.2: Travel distribution by class label



As shown in table 2.2, each travel has n frames. Each of these frames is associated with a label as well as the trip itself. As a trip may have variable number of frame depending on its duration, it could be interesting to know the classes' distribution in terms of frames :

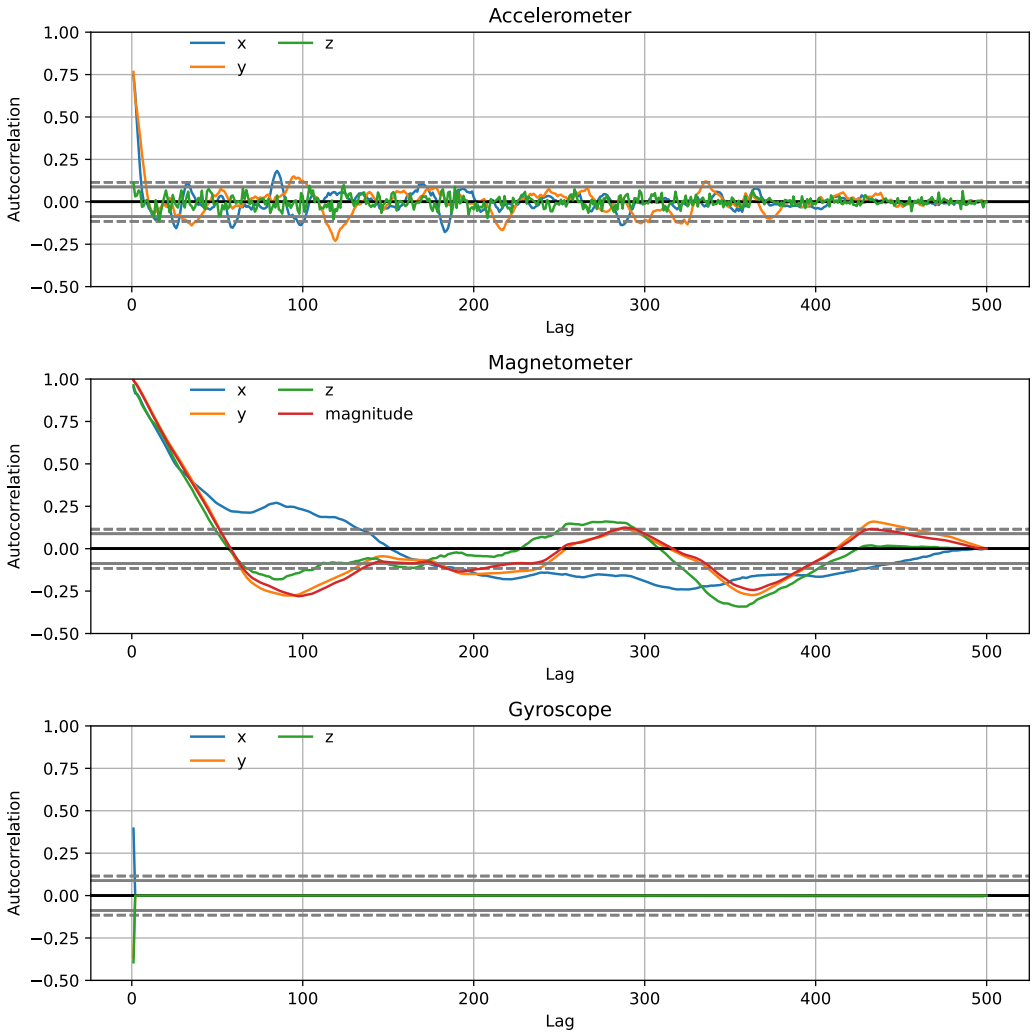
Figure 2.3: Frames distribution by class



The Figure 2.3 shows similar results as Figure 2.2. However, it seems that class distribution is more balanced at a frames level, with 47.27% of positive frames and 52.73% of negative frames. Despite this little difference, we may consider that regardless of granularity level, the labels are balanced and allow us to work either with trips or frames directly.

To further understand the data, the autocorrelation coefficient can be used to analyze the relationship between the data. It is a metric that is often used in time-series analysis and measures how much a time-series is linearly related to the lagged version of itself [27]. It allows to know if a time-series is identically and independently distributed (iid) [28]. This is crucial information as in the machine learning domain, the application of regression models implies a common hypothesis that assumes the data are iid [27]. Besides knowing the distribution, autocorrelation may help to find some seasonality and trends in time-series data [27]. The figure 2.4 shows the autocorrelation's coefficient for each axis of each sensor with itself:

Figure 2.4: Sensor's autocorrelation coefficient



The y-axis (vertical) units go from -0.5 to 1 and represent the autocorrelation's coefficient. Despite y-axis plots on Figure 2.4 going from -0.5 to +1, the resulting output of autocorrelation's coefficient can range from -1 to +1. The x-axis (horizontal) units represent the number of past values in the time-series and were arbitrarily chosen between 0 and 500. As an example, a positive autocorrelation value of 1 with a lag value of 5 noted i, assumes that if a $i - 5$ value in the time-series increased, the $[i - 4, i]$ values would increase proportionally. On the other hand, a negative value of 1 with the same lag value assumes that a $i - 5$ value increase results in a proportionate decrease in the $[i - 4, i]$ time-series values. An autocorrelation value of zero indicates no correlation between the i lag and the current time-series value.

In Figure 2.4, continuous gray lines represent a confidence interval of 99% and the dashed ones, represent a confidence interval of 95%. In this Figure, despite that the magnetometer's autocorrela-

tion coefficient takes more lag steps to decrease, both the accelerometer's and the magnetometer's autocorrelation coefficient decrease as the lag steps increase and tend towards zero. At first glance, the gyroscope shows to be in the same case of magnetometer and accelerometer with a rapid decrease in the autocorrelation coefficient. However, the data used to plot the Figure 2.4, sampled in Table 2.2, shows there are no real variations in all gyroscope axes over time. It is probably the reason why the autocorrelation value of the gyroscope is near zero. The general hypothesis given by Figure 2.4 is that the correlation of the time-series values of each sensor is insignificant from lag 50.

Figure 2.5 shows a normalized sample of the first 500 values for each sensor. The units depend on each sensor. Each plot has 4 curves representing x, y, and z axes and the magnitude between these axes. As a reminder, the vector magnitude also called the norm, where a is the time-series in our case, i is the index value and n is the last value of the time-series, is designed by Equation 2.2 [29]:

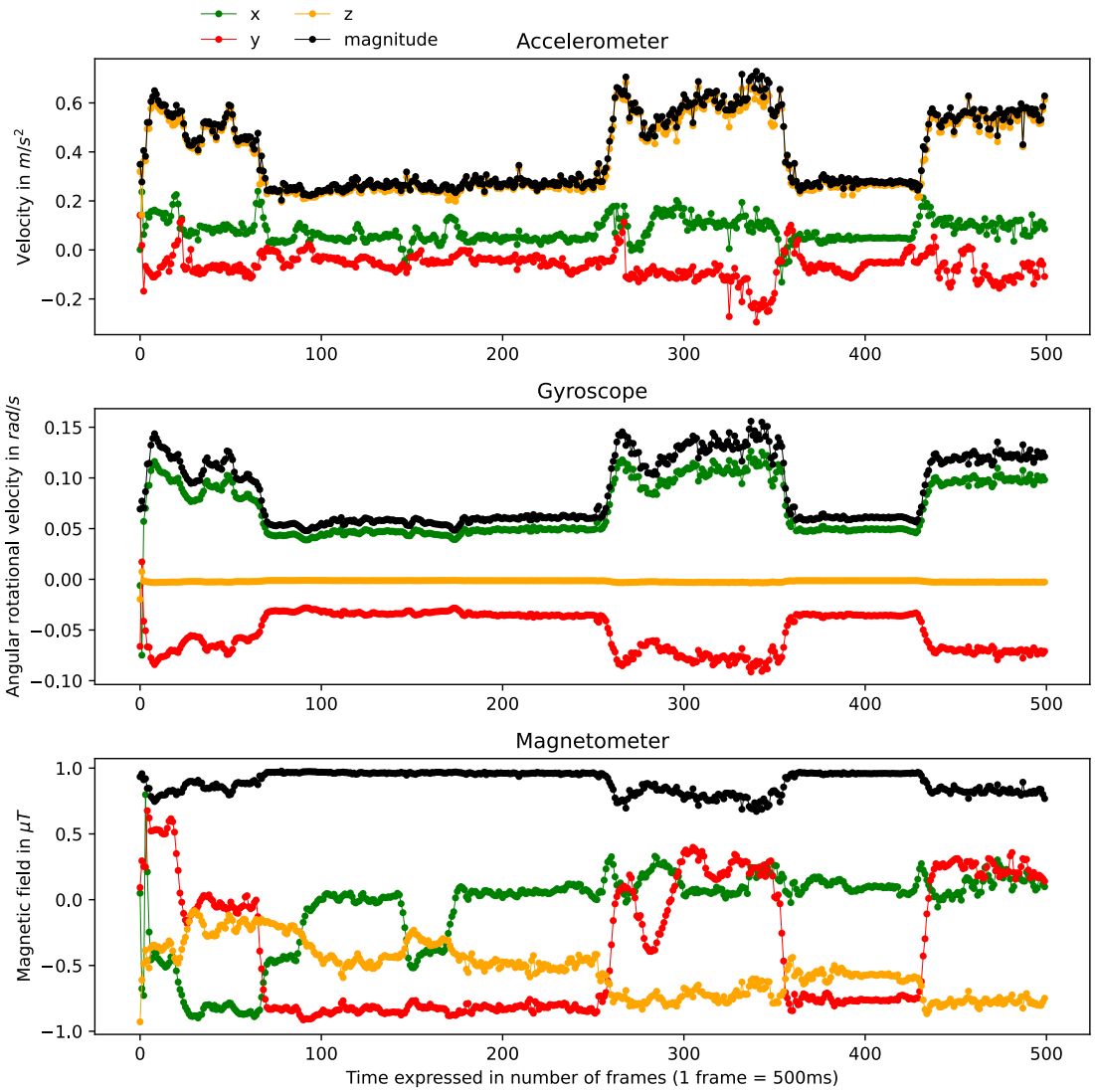
$$||a|| = \sqrt{a_1^2 + a_2^2 + ... + a_n^2} = \sqrt{\sum_{i=1}^{n} a_i^2}. \tag{2.2}$$

Thus, the plotted magnitude m in Figure 2.5 is calculated between x, y and z axes for each sensor and can be expressed by Equation 2.3:

$$||m|| = \sqrt{\sum_{i=1}^{n} x_i^2 + \sum_{i=1}^{n} y_i^2 + \sum_{i=1}^{n} z_i^2}. \tag{2.3}$$

The magnitude represents the norm of a vector and measures its length among all representations of this vector. In Figure 2.5, the black curve represents the distance between the three axes for each sensor and allows us to compare directly this magnitude between the sensors instead of comparing each axis independently. This figure shows the correlation between the magnitude of the accelerometer and the gyroscope. Both sensors have similar behavior, especially for the peaks and regularity of the magnitude.

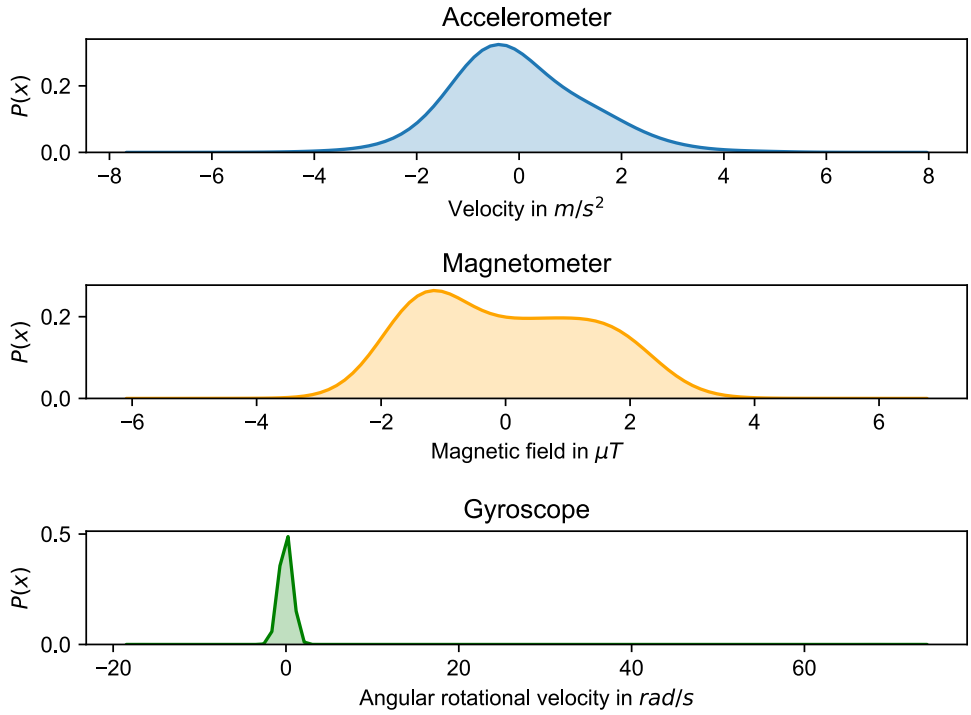
Figure 2.5: Sensor's sample normalised



Moreover, it would be relevant to know the distribution and how the data are spread. One way to do this is to compute each sensor's time-series' mean and standard deviation and compare them together. The other way, which would be visually more practical, is to plot the density, such as in Figure 2.6. The Principle Component Analysis (PCA) has been used to plot each sensor with only one distribution curve instead of plotting the three-axis of each sensor. PCA may be used for dimensionality reduction purposes, for example. In Figure 2.6, the vertical axis represents the probability of the horizontal axis values. Accelerometer and gyroscope have both a unimodal distribution, which results in a values interval being particularly dominant, among others. On the other hand, the magnetometer has a bimodal distribution. Another indication given by Figure 2.6 is that the distribution of accelerometer and magnetometer are wider than the gyroscope. It could be explained by less rotational forces

applied on gyroscope than velocity forces applied on accelerometer and magnetic field applied on magnetometer in this trip.

Figure 2.6: Sensors probability density distribution



2.2 Data gathering tools

In this work, a data set that matches the classification problem has not been found. A way of collecting sensor's data directly on a smartphone while people were carpooling had to be created. The chosen solution was to create a basic mobile application with Ionic v.5.4.16, a framework that allows developing hybrid mobile applications that can be used on both Android and iOS.

The server-side of the application uses Firebase as a database to store NoSQL data collected from the mobile sensors.

In the first data collection phase, the application has been used only by me and some friends, to be able to control the recording environment and to avoid collecting noisy data. In a second phase, which will be carried out after this work, the application will be distributed to Mobilidée's employees to collect more data. The main functionalities of the application are :

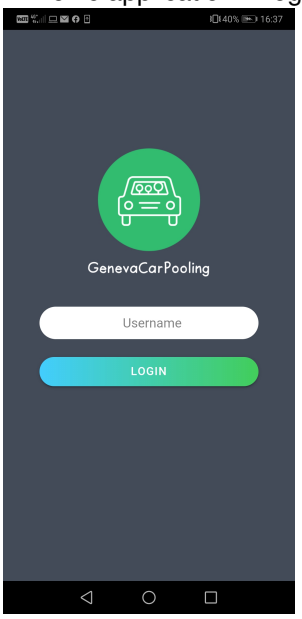
- Login,

- Create a trip,
- Invite passenger(s),
- Start recording sensor's data,
- Stop recording and send data to server.

Login

This had to be as simple as possible to let users type only their username to be connected. There is not any password, meaning a person takes the identity of another person. This is why the application needs to be used in a controlled environment, but there is no personal information stored in the application, other than the username. The main reason to create a login, in this case, is to list all potential passengers and be able to send them notifications. Here is the login page of the application:

Figure 2.7: Ionic application - Login page

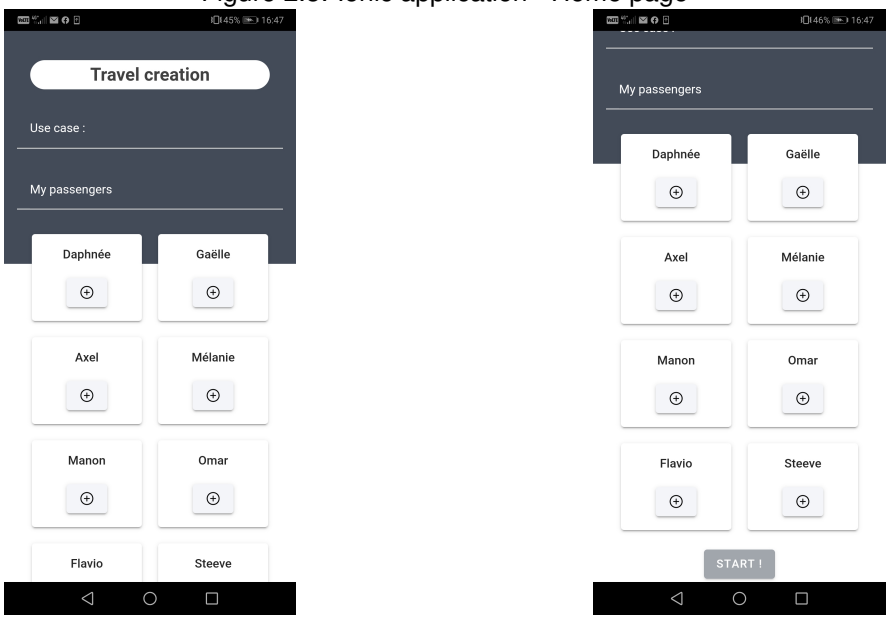


The login page contains only one input field where the user must enter his username and a button, which allows him to connect. The available usernames are pre-configured in the Firebase administrative panel. Once a user has logged in, a notification token id is associated with the smartphone and is stored in Firebase backend.

Create trip

Once the user is logged in, there is a redirection to the home page. This page, illustrated in Figure 2.8, lets users create a trip by choosing the travel use case and by inviting passenger(s), which are explained and listed in the section 2.3. The only person on a trip who must fill in this page is the driver.

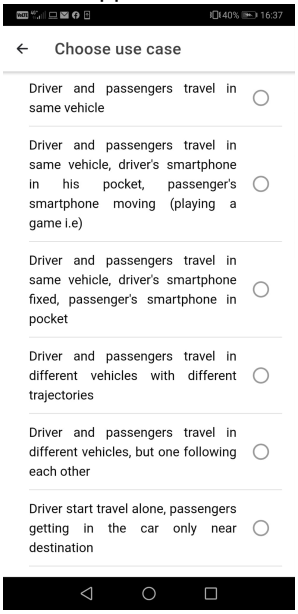
Figure 2.8: Ionic application - Home page



Inviting passenger(s)

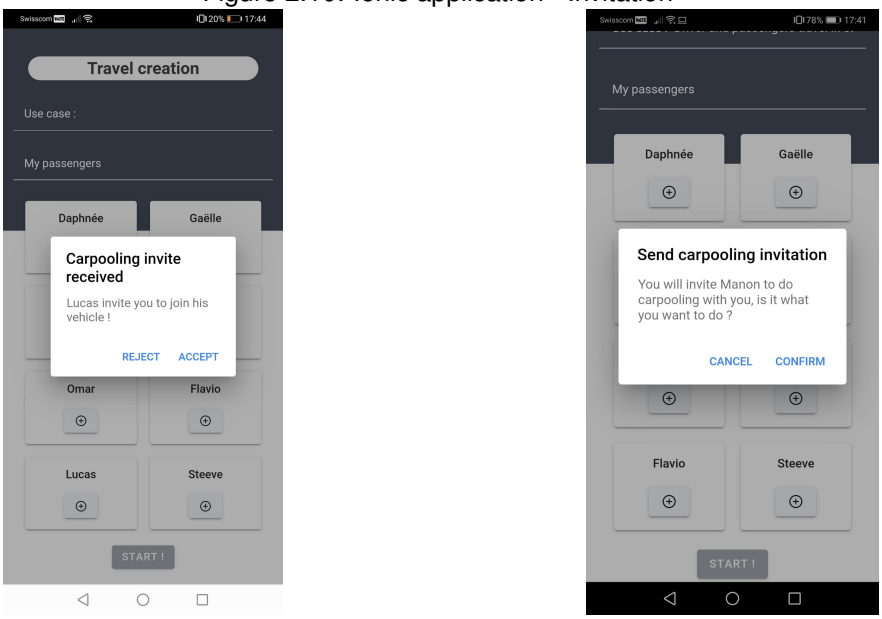
Once the driver has chosen the correct use case for the trip in the page showed in Figure 2.9:

Figure 2.9: Ionic application - Use case page



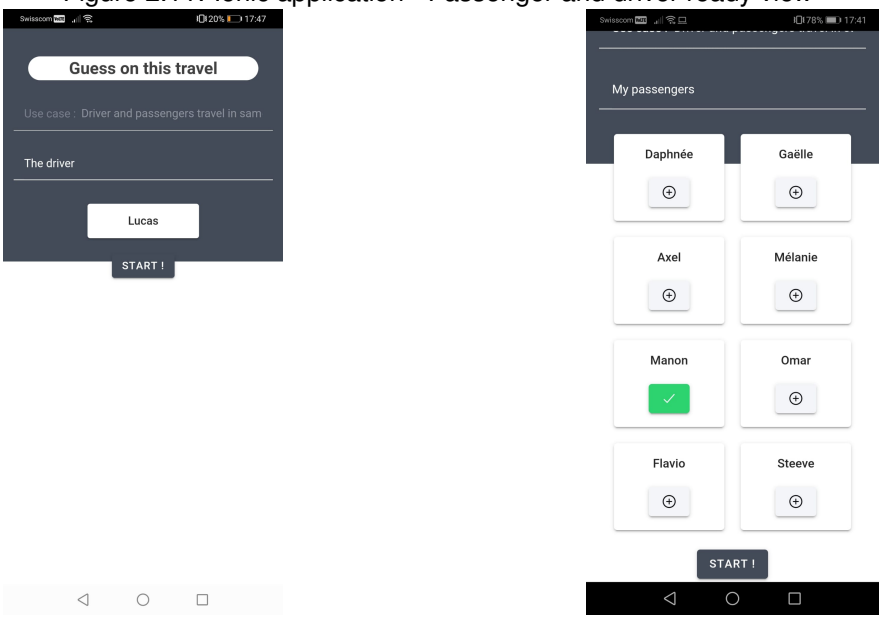
The driver must invite one or multiple people listed on the home page to join his trip in order to be able to begin recording the sensor's data. When the driver selects a passenger, a popup will be displayed to ask his confirmation, and once it is done, the notification goes directly on the smartphone of the selected passenger. The left side of Figure 2.10 is the passenger's view, and the right side is the driver's view.

Figure 2.10: Ionic application - Invitation



Once the passenger has accepted the invitation, he will be redirected to the view showed on the left side of the Figure 2.11 and the driver's view will be updated as the right side of the same figure.

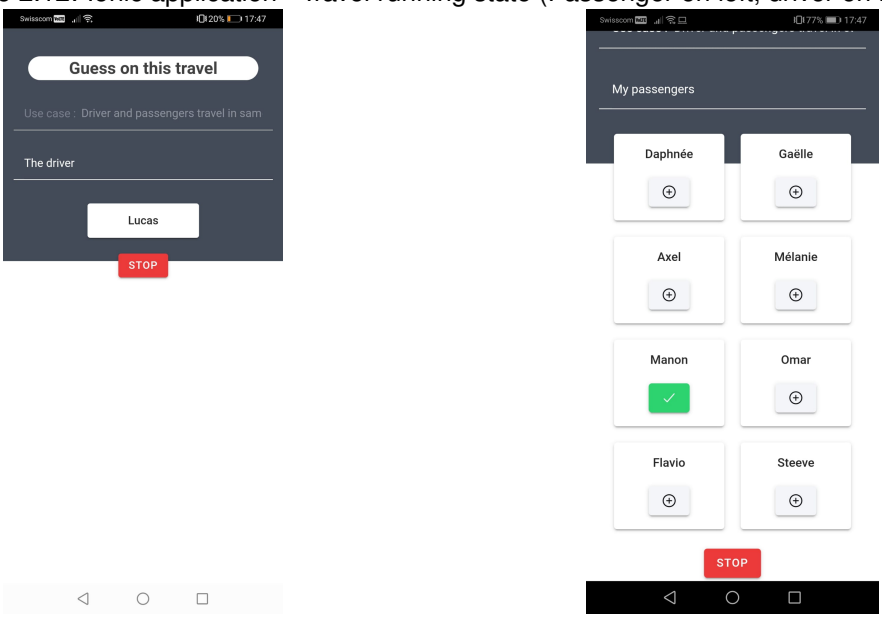
Figure 2.11: Ionic application - Passenger and driver ready view



Start recording sensor's data

The travel is now ready to start with one driver and one passenger. Both driver and passenger must tap on the "Start" button to begin mobile's sensors recording. The first time the users start a carpooling trip on the application, a confirmation will be asked to collect sensors data, according to the GDPR. During the trip, sensors data will be recorded every 500 milliseconds. This was an arbitrary choice to try to capture as many variations as possible in the magnetometer, accelerometer, and gyroscope without consuming too much battery. Too small of an interval could cause the battery to run low. The application's state while travel is running is shown in Figure 2.12.

Figure 2.12: Ionic application - Travel running state (Passenger on left, driver on right)



Stop recording data

Finally, both driver and passenger has to press the button "Stop" illustrated in Figure 2.12 to conclude the carpooling trip. It will send their raw data to the firestore, a local version of firebase that allows data to be temporarily stored on the mobile if it has no internet connection; otherwise, it sends them directly to the database. The application's state is reset to its initial state, as shown in Figure 2.8.

2.3 Use cases

The use cases treated in this section refer to possible scenarios that could happen in a carpooling trip. These scenarios have been discussed with Mobilidée and the thesis supervisor to cover a large scope of cases that could happen in reality. As shown in section 2.1, the mobile sensor's axes are fixed, so it has been decided to use different smartphone positions and variate people's activity during the carpooling travel. The use cases, briefly shown in Figure 2.9, are the following:

1. Driver and passengers travel in the same vehicle, smartphones at rest.

2. Driver and passengers travel in same vehicle, driver's smartphone in his pocket, passenger's smartphone in movement (playing a game i.e).
3. Driver and passengers travel in same vehicle, driver's smartphone fixed, passenger's smartphone in pocket.
4. Driver and passengers travel in different vehicles with different trajectories, smartphones at rest.
5. Driver and passengers travel in different vehicles, but one following each other, smartphones at rest.
6. Driver start travel alone, passengers getting in the car only near destination, smartphone at rest.

Only the use case number 1 and 4 cited above would be considered for the data collection made in this thesis. Furthermore, I am the only one who collects the data with multiple smartphones. These decisions were made to be able to keep data collection under control while minimizing the risk of noisy data wrongly recorded by users, for example. A first data collection was made, introduced in section 2.1, to be able to train statistical models. A second data collection in a more real environment would be done after this thesis to test the trained models. In this second phase, most of the use cases cited above would be considered, and the data collection application would be distributed to most Mobilidée employees. The second phase will not be discussed in this thesis.

2.4 Models

2.4.1 General information

This section will discuss the different statistical models used in the classification and regression problems, as the scope of this thesis will approach both. Machine learning models might be represented as a set of weights applied on a dataset to give a prediction. Depending on the task to solve, prediction can be discrete and denoted as classification, or they can be continuous and would be denoted as regression. Then, the model's objective is to train on a data set to minimize a loss function (represents the error of the model), noted L, by changing these weights, noted W. In statistics, the loss function is used for parameter optimization such as W, and consists of a difference between expected output and model's output.

2.4.2 Classification

2.4.2.1 Linear regression to logistic regression

As a reminder of section 1.1, the classification model architectures discussed here are related to carpooling classification. One of the most simple statistical models used in machine learning is the

linear regression algorithm. This model makes a linear relationship assumption between a scalar prediction, noted y, and one or more explanatory variables, noted x. Thus, linear regression uses as many parameters, noted β, as explanatory variables, also called features, and chooses those who minimize the prediction error. Equation 2.4a illustrates the linear regression model applied on each dataset instance where y_i is the prediction for the i^{th} instance, β_0 is the bias, n is the number of features and so the x_{ij} is the j^{th} feature for the i^{th} instance. Moreover, the linear regression might be written as the matrix approach, as shown in equation 2.4b, where W is the β matrix, or more generally, the weights matrix and X is the features matrix.

$$y_i = \sum_{j=1}^{n} \beta_j \cdot x_{ij} + \beta_0, \tag{2.4a}$$

$$y = W^T X. \tag{2.4b}$$

Once the model has obtained the predictions y, its' objective will be to choose the best W to minimize the loss function. Mean squared error (MSE) is a used loss function in linear regression and is expressed as follow:

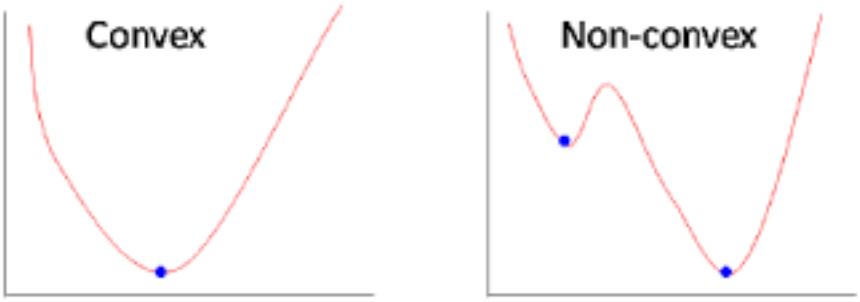
$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \tag{2.5}$$

MSE will calculate the squared average difference between the model output $\hat{y}$ and the correct label y expected, which must be minimized to obtain the most accurate model. Two approaches could compute this minimization:

- Analytical solution,
- Gradient descent.

The analytical solution exists because the MSE in linear regression is a convex function, which means there is only one minimum, see the left side of Figure 2.13. This minimum might be found by using the partial derivatives of MSE with respect to w and with solving the equation by finding the only one existing 0.

Figure 2.13: Convex vs non-convex function



[30]

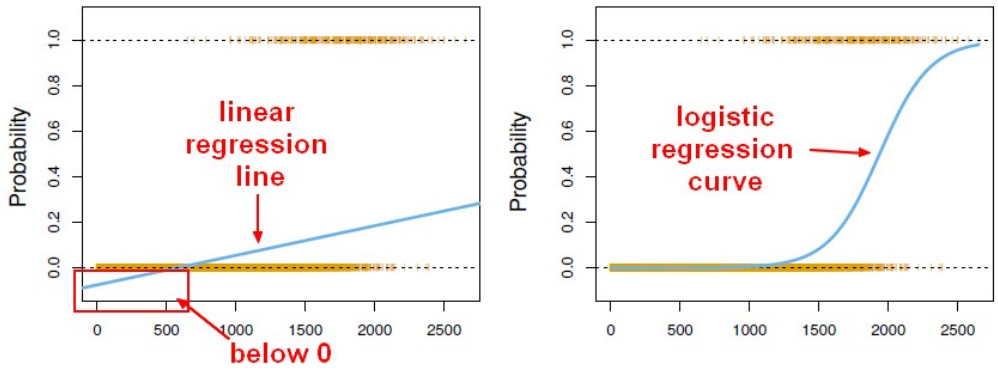
As the matrix analytical solution showed in Equation 2.6 requires a matrix inversion calculation which his cost growth as the number of features increase, gradient descent is commonly preferred.

$$\beta = (X^T X)^{-1} X^T Y. \tag{2.6}$$

The gradient descent part is a fundamental concept to keep in mind as it will be used in many machine learning loss minimization problems. It is an optimization algorithm that iteratively moves in the steepest direction of a function by taking its' negative gradient. The gradient is a term to designate a vector in which every position is a partial derivative of a multi-variate function. Hence, as the derivative of a function gives the direction of the highest increase, taking the negative gradient will follow the fastest direction that minimizes the function output.

The usage of linear regression is limited, especially in binary classification, such as in carpooling classification discussed in this thesis. Indeed, the outputs produced by the linear regression could be any real number. This approach does not allow for probabilistic analysis. However, negative values could represent one class and positive values in another class. The problem with this technique is for optimizing the statistical model since the outputs are not normalized between 0 and 1, for example. Figure 2.14 shows another algorithm that can address this issue, the logistic regression while showing the difference between linear regression and logistic regression [31].

Figure 2.14: Linear regression line vs Logistic regression curve



[31]

The solution is to use a logistic function, such as sigmoid, on linear regression model predictions to obtain probabilities between 0 and 1. See the logistic application on linear regression output data on right side of Figure 2.14. This model is called logistic regression, and its predictions could be obtained by the Equation 2.7 where σ is the logistic function.

$$y = \sigma(W^T X + b). \tag{2.7}$$

Instead of minimizing the MSE as in linear regression, the logistic regression parameters optimization will choose the parameters to maximize the probability. This is known as the maximum likelihood estimation (MLE). The MLE's general intuition is to maximize the conditional probability of observing the data given probability distribution of X and its parameters θ:

$$max L(X|\theta). \tag{2.8}$$

As X is the joint probability distribution of all observations, Equation 2.8 could be written as the sum of the conditional probability, as shown in Equation 2.9. Moreover, the logarithm function is commonly used to avoid unstable numerical issues caused by multiplying multiple probabilities together. Furthermore, using the logarithm function on another function will not change its' argmax.

$$max \sum_{i=1}^{n} log(P(x_i|\theta)). \tag{2.9}$$

In the literature, it is commonly known as the log-likelihood function. In supervised learning, by following the underlying logic of Equation 2.8, the problem could be expressed as the conditional

probability of an output y given the inputs X:

$$P(y|X). \tag{2.10}$$

Thus the definition of maximum likelihood in supervised learning can be rewritten as follow:

$$max \sum_{i=1}^{n} log(P(y_i|x_i; Q)). \tag{2.11}$$

where Q is the logistic regression model. To use maximum likelihood function, a probability distribution needs to be assumed. The binomial probability distribution is the most often used in classification problem with two classes [32], which is defined by the following formula:

$$p \cdot k + (1-p)(1-k) \; for \; k \; \in \{0, 1\}. \tag{2.12}$$

where p is the probability, and k is the possible outcomes. Hence, this probability distribution can be extended to the logistic regression model where the probability $\sigma(W^T X)$ is the output of the model, and the possible outcome y is the correct label given by the data set, as illustrated in Equation 2.13.

$$L = \sigma(W^T X) * y + (1 - \sigma(W^T X)) * (1 - y). \tag{2.13}$$

The likelihood function needs to be maximized and computed for all instances in the data set, where n is the number of instances. Please, see the equation 2.14, which also uses the logarithm function to avoid numerical issues, as discussed previously in this section.

$$L = argmax \frac{1}{n} \sum_{i=1}^{n} log(\sigma(W^T x_i)) * y_i + log(1 - \sigma(W^T x_i)) * (1 - y_i). \tag{2.14}$$

Besides, it is common to minimize the loss function in machine learning optimization problems rather than maximize it. Hence, the negative of the log-likelihood function minimization instead of the log-likelihood maximization function is used, as shown in Equation 2.15

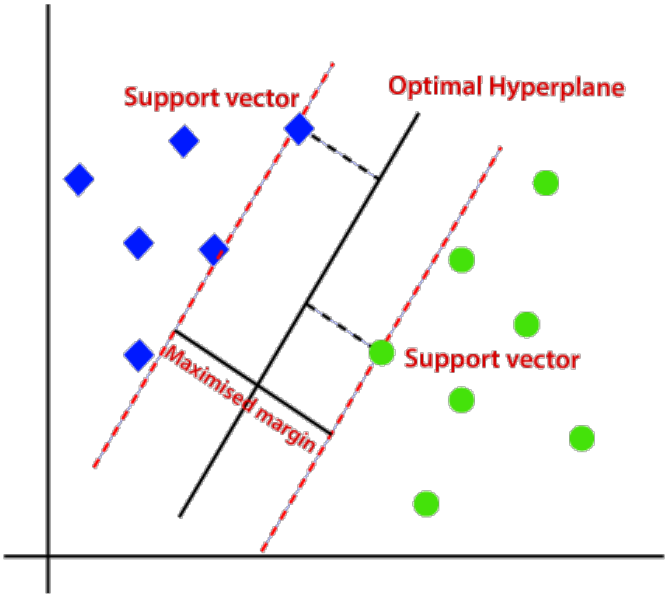
$$L = argmin \frac{1}{n} \sum_{i=1}^{n} -(log(\sigma(W^T x_i)) * y_i + log(1 - \sigma(W^T x_i)) * (1 - y_i)). \tag{2.15}$$

The Equation 2.15 is in fact the binary cross-entropy loss. As the MSE discussed above in the linear regression, cross-entropy is used to compute an average error between the predicted label and the true label. As the cross-entropy is not a convex function, an iterative optimizing algorithm, such as the gradient descent explained above, must be used to find the best logistic regression weights W.

2.4.2.2 Support vector machine

Support vector machine (SVM) is a supervised machine learning algorithm initially designed for two-group classification problems[33]. SVM aims to create a decision boundary, also called hyperplane, based on sample data that allows separating classes in space to classify other instances of data efficiently. Two groups of SVM can be found: the one who is used on linearly separable data and the one on non-linearly separable data. In the linear case, the SVM tries to find the hyperplane that separates each class while maximizing the hyperplane and support vector's margin. Support vectors are found by using the closest point of each class to the hyperplane, and the margin is the distance between the hyperplane and the support vectors, see Figure 2.15 where each color and shape corresponds to a different class.

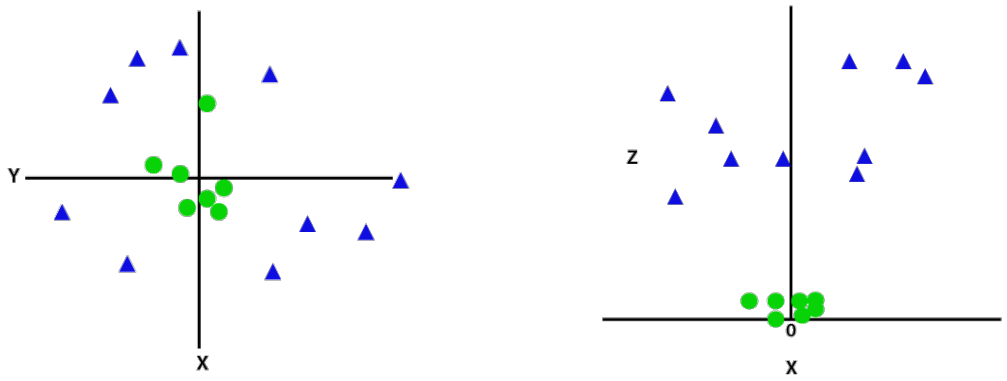
Figure 2.15: SVM - Linearly separable data - Decision surface



[34]

In non-linearly separable data, a straight line, as in Figure 2.15, cannot separate the data into two distinct classes. Considering the left side of the Figure 2.16 where data from two classes are shown in two different colors, SVM will create a third dimension to try to make a clear separation between classes as shown in the right side of the Figure 2.16.

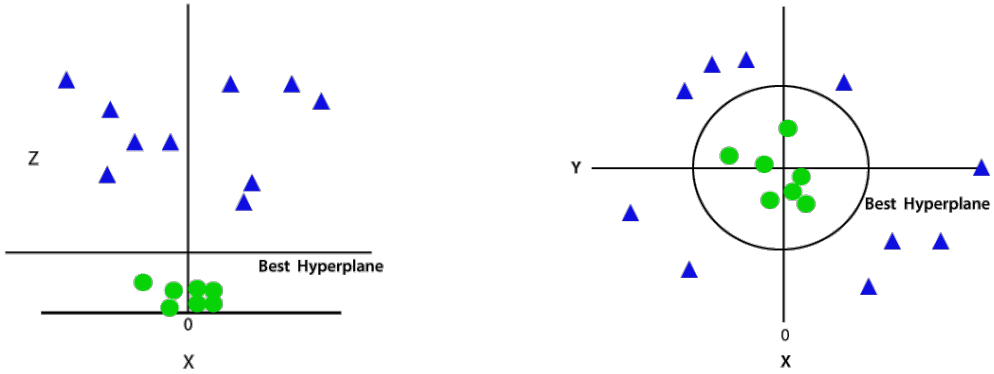
Figure 2.16: SVM - Non-linearly separable data - Create new dimension



Then, SVM will create a hyperplane to separate the data into classes in the three-dimensional representation shown in the left side of Figure 2.17 and could be seen in two dimensions by converting the data back to its original representation and see how the hyperplane looks like in the right side of the Figure 2.17. The third dimension z here is computed the Equation 2.16 where x and y are two features of the data set.

$$z = x^2 + y^2 \tag{2.16}$$

Figure 2.17: SVM - Non-linearly separable data - Hyperplane decision



SVM's technique when dealing with non-linearly separable data is to find the best function to map data from one space to another space, which is more straightforward for SVM to manage. It is common to use kernel functions, such as Radial basis function (RBF), polynomial kernels, and Gaussian kernels, to reduce the complexity of finding the mapping function. Assuming the data to analyze are vectors, a simple kernel example that respects kernels' fundamental property would be the inner product of two vectors [35]. See the Equation 2.17 where x and y are two dataset feature vectors and n is the number of instances.

$$k(x, y) = x^T y = \sum_{i=1}^{n} x_i y_i. \tag{2.17}$$

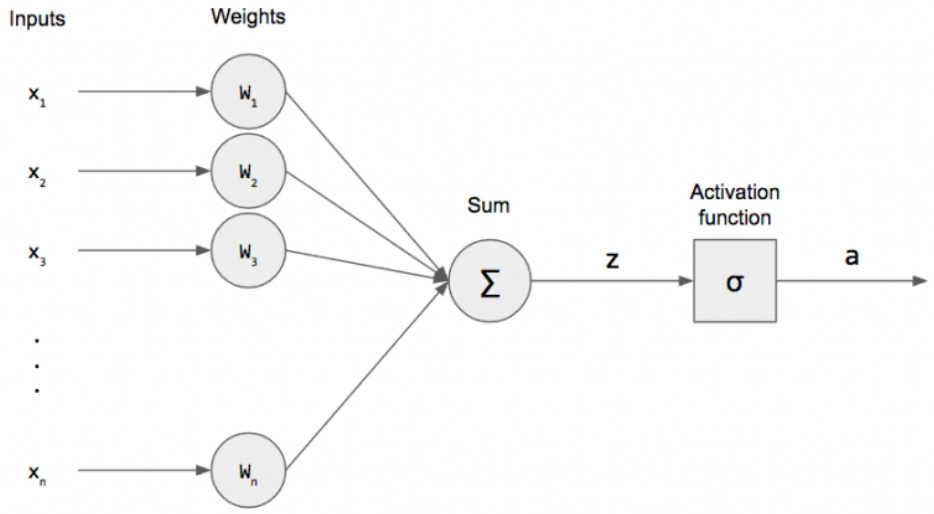
This kernel function is usually called the linear kernel. Much more complex kernel methods, as cited above, are often used in SVM; however, we will not discuss them in more detail in this thesis.

2.4.2.3 Artificial neural networks (ANN)

Perceptron

The Perceptron, developed in 1958 by F. Rosenblat, is the simplest example of neural network architecture and was designed with the main idea of having some of the fundamental properties of the human brain system [36]. Indeed, the Perceptron architecture contains only one neuron layer with one single neuron, as shown in Figure 2.18, where a are the predictions, the "Sum" and the "Activation function" units are parts of the Perceptron neuron.

Figure 2.18: Perceptron



[37]

The Equation 2.18 gives the mathematical way to get a prediction for the k^{th} instance from a Perceptron where n is the number of instances in the training data, b is the bias that controls the y-intercept, and σ is a non-linear activation function such as ReLu, Sigmoid or Tahn.

$$y_k = \sigma(\sum_{i=1}^{n} w_i \cdot x_{k,i} + b) = \sigma(W^t X + b). \tag{2.18}$$

Assuming the activation function is a Sigmoid in above Equation, the Perceptron will make similar predictions as logistic regression, see Equation 2.7. The difference lies in the Perceptron not assuming a Bernoulli distribution and is more flexible with the possible activation functions, such as ReLU instead of Sigmoid. Moreover, the Perceptron only penalizes instances that are wrongly classified. Thus, the Perceptron loss function is defined by equation 2.19, where M is the set of misclassified instances.

$$- \sum_{x_i \in M} (w^T x_i + b) y_i. \tag{2.19}$$

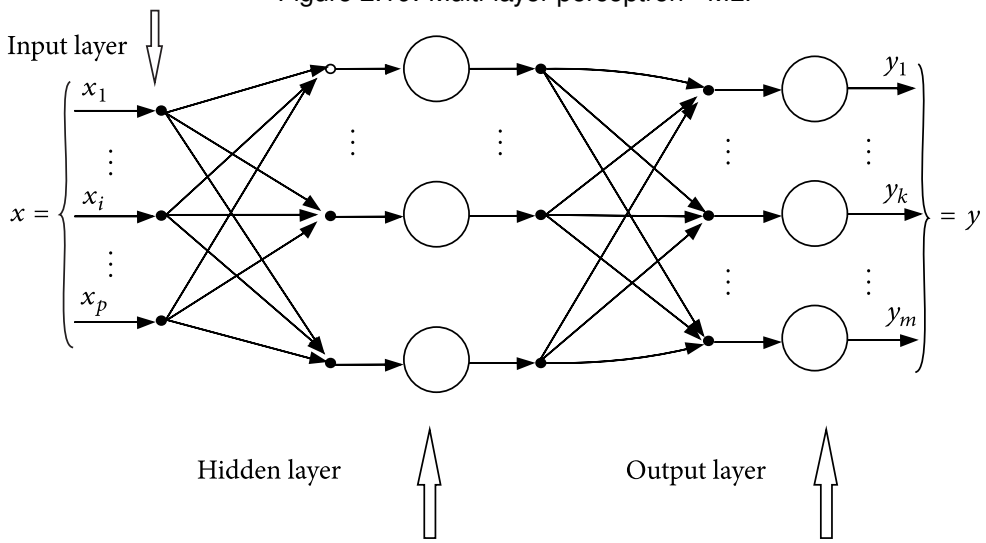
Then, the Perceptron's loss function minimization needs to be computed with the gradient descent to find the optimal weights. The gradient of the loss function is then defined by Equation 2.20.

$$\nabla L(w) = - \sum_{x_i \in M} x_i y_i. \tag{2.20}$$

Multi-layer perceptron (MLP)

The Perceptron can only solve linear problems while the multi-layer perceptron has not this limitation, which is a type of feedforward artificial neural network. The term feedforward designs the fact that the connections between the nodes, inside the network, do not form a cycle. MLP architecture consists of having an input layer, an output layer, and at least one hidden layer. The hidden layer is composed of multiple perceptron units; see Figure 2.19. MLP is a fully-connected network where each node in a specific layer is connected with a certain weight w_{ij} to every node in the following layer.

Figure 2.19: Multi-layer perceptron - MLP



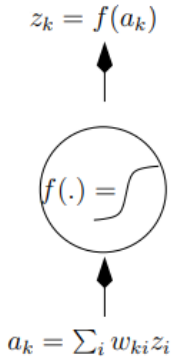
[38]

The dataset inputs size gives the input layer size, and the output layer has an activation function that

maps the hidden layer results to the desired output. In a multiple label classification problem, the softmax activation function could be used as the output non-linear function to map the hidden layer results to a probability to belong in each class. The main difference between sigmoid and softmax non-linear function lies in the output properties. Either Sigmoid and Softmax outputs lie between 0 and 1; however, the sum of the softmax outputs always gives 1 and can be interpreted as probabilities.

Any perceptron unit in any hidden layer will always have the same operation: receiving input a_k from the previous layer which is given by the weighted combination of the outputs of the previous layer and weights w_k of the current layer where current unit k applies a non-linear activation function f. See Figure 2.20 where a hidden unit k receives activation value a_k, the output from the previous layer z and use the weights w_k of the current unit k.

Figure 2.20: MLP - hidden unit k



Similar to the hidden layers, each k output unit gets as activation the linear combination of the previous layer output with the output weights. Hence, each k unit in an MLP has a set of j weights representing a weights matrix of $j \cdot k$ parameters to optimize. For example, assuming a dataset of instances with 200 features each using an MLP with one hidden layer of 50 units, there will be $50 \times 200 = 10000$ parameters to train only for the hidden layer. To be able to learn all these parameters, the backpropagation of the error with respect to the weights w must be computed. This notion refers to the partial derivative of the loss function with respect to any weights in the MLP. The intuition behind this calculation tells how quickly the error changes when the weights change[39]. Before going further, an essential notion is the chain rule; see Equation 2.21. It is used when the derivative of a composite function needs to be computed [40].

$$(f(g(x)))' = f'(g(x)) \cdot g'(x) \equiv \frac{\partial dz}{\partial dx} = \frac{\partial dz}{\partial dy} \cdot \frac{\partial dy}{\partial dx}. \tag{2.21}$$

In Equation 2.21, dz refers to $f(g(x))'$, dy to $g'(x)$ and dx to $g(x)$. Indeed, the final output of an MLP is

a composition of multiple functions. Thus, to backpropagate the error through the weights of the MLP, the partial derivatives using the chain rule will be used. Let's assume a loss function $L(w)$, which is usually the cross-entropy in a binary classification problem, computed on the MLP outputs shown in Figure 2.19, which has to be minimized with respect to the weights w. By doing only the first partial derivative step from the output to the hidden layer, assuming w contains only weights from j to i unit links between the output layer and the hidden layer. The contribution of a w_{ij} weight to the L comes only through the activation level a_i which is given by Equation 2.22:

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_i} \cdot \frac{\partial a_i}{\partial w_{ij}}. \tag{2.22}$$

Equation 2.22 gives the gradient of the error with respect to w_{ij} which connects the output of unit j to the input of unit i where a terminology simplification can be done as:

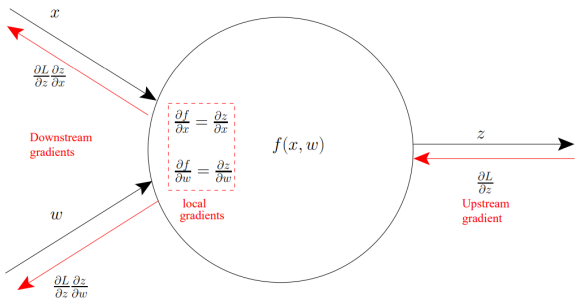
$$\frac{\partial L}{\partial a_i} = \delta_i, \tag{2.23a}$$

$$\frac{\partial a_i}{\partial w_{ij}} = z_j, \tag{2.23b}$$

$$\frac{\partial L}{\partial w_{ij}} = \delta_i z_j. \tag{2.23c}$$

Thus, this gradient is used as the weight update rule for each hidden layer in any MLP as it may contain multiple hidden layer. More generally, the weights update rule inside a unit may be expressed as $\frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w}$ for one hidden layer, as shown in Figure 2.21.

Figure 2.21: MLP - General weights unit update rule



In Figure 2.21, the upstream gradient refers to the gradient computed with the output of the next node

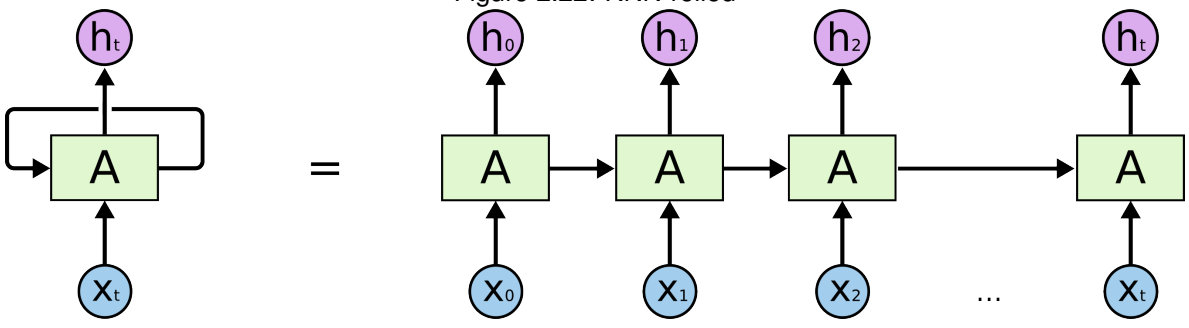
with respect to its input. The local gradients refer to the gradient of the current node output with respect to the current input. The downstream gradients refer to applying the chain rule by multiplying the upstream gradient with the local gradients and passing them through the previous node in the network to continue backpropagation of the error.

In Figure 2.21, two downstream gradients are computed : one with respect to the weight parameter w and another with respect to the input x. Assuming that x is the input given to the network, the error gradient with respect to it does not need to be calculated. However, assuming that x is given by another hidden unit, its gradient must be computed to continue passing backward the propagation of the error since the input layer where only the gradient with respect to w has to be done.

Reccurent neural networks (RNN)

RNN is a type of artificial neural network where previous outputs are used to predict further outputs. The RNN assumption is that a single input item in a given series is related to others as it uses the information of previous input to help him taking predictions on further ones. In other words, the RNN decision on an instance is influenced by what it has learned from its past features, in addition to the other instances' influence on the network in general. A known advantage of RNN among feedforward neural networks is that they are designed to allow dynamic vector sizes as input. An important notion here is the timestep, a common term used in RNN about specific variable states at a given time in the network, noted t. Furthermore, the RNN architecture has a single hidden layer, which takes as input the data x^t and the previous output made on data to give an output at the current timestep, see Figure 2.22. Colah's quote may exemplify an analogy between how RNN works and how Human understands the natural language: "You understand each word based on your understanding of previous words. You don't throw everything away and start thinking from scratch again. Your thoughts have persistence" [41].

Figure 2.22: RNN-rolled



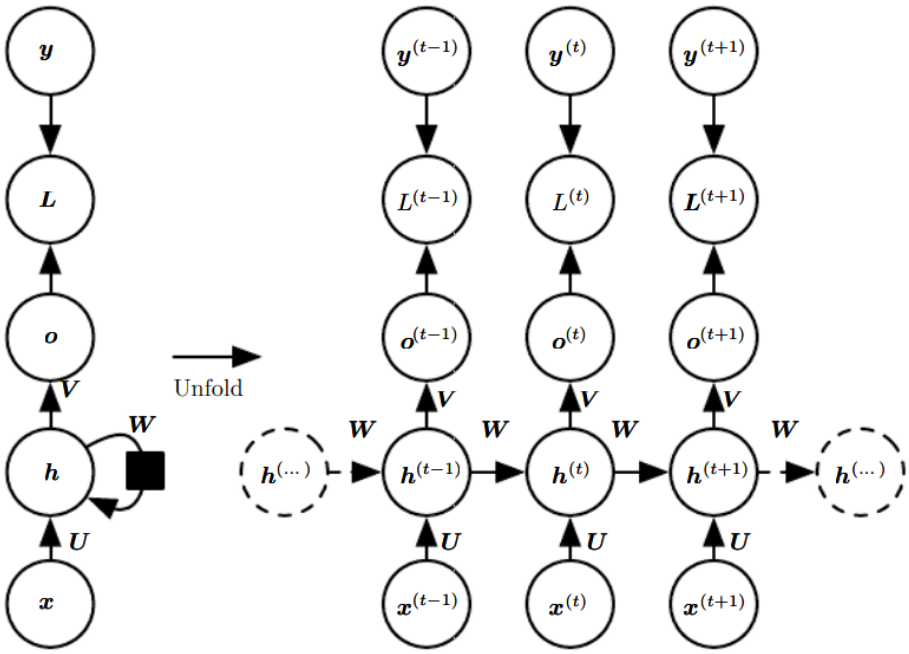
[41]

The RNN illustrated in Figure 2.22 shows two ways of representing how it works. On the left side, it shows the RNN, commonly known as the rolled representation, at a given timestep t where the hidden layer A taking some input x_t and outputs a value h_t where the arrow recursively pointing on A represents the information passed to the $t+1$ timestep. On the right side of the Figure 2.22, it shows the RNN, commonly known as the unrolled representation, at some multiple timestep from 0 to t. The information passed through the timesteps is more obvious in this representation where each x_t input gives an output h_t and let information to the next timestep $t+1$. This can be formulate by Equation 2.24 where $h^{(t)}$ is a function f of $h^{(t-1)}$ and $x^{(t)}$ where θ are the parameters of f.

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}, \theta). \tag{2.24}$$

The RNN can also be expressed with a computational graph that maps an input sequence x to a corresponding sequence of output o, commonly called many-to-many, see Figure 2.23. A computational graph is a directed graph where each node refers to operations or variables.

Figure 2.23: RNN - Many-to-many - Computational graph



[42]

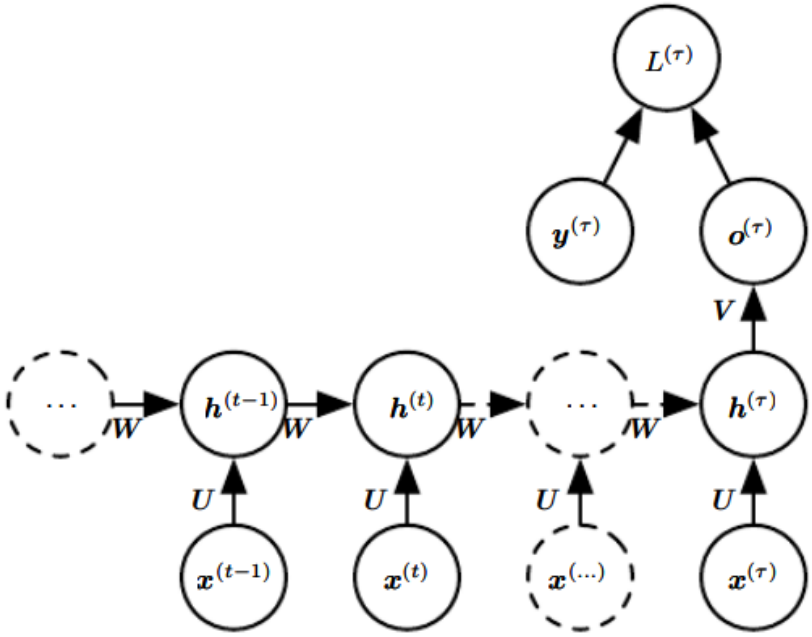
In Figure 2.23, the loss function L measures the difference between the model prediction o and the target value y. A weights matrix of U parametrizes connections between x and the hidden layer while the hidden layer is parametrized with a weights matrix of W by itself through timesteps of t, and the

hidden layer to the output layer is parametrized with a weights matrix of V.

One may use this architecture to generate text. At training time, it will work like a feature extractor by trying to predict the next word in a given sentence where each component of x would be a word given to the RNN that would try to find the next word of y_t by predicting o_t at a given timestep of t. This architecture uses a little trick, such as when testing time, the model generates a sentence, word after word, by passing through the previous word predicted to help it generate the next one and output the final tense without any input.

This leads to another architecture used in this thesis, where the RNN maps an input sequence of x to only one output of o. This is commonly referred as many-to-one RNN architecture and can also be shown as a computational graph, see Figure 2.24 which uses the same notation as Figure 2.23 except τ which represents the last step in the x input sequence.

Figure 2.24: RNN - Many-to-one - Computational graph



[42]

Hence, the Equations 2.25a, 2.25b, 2.25c, 2.25d mathematically expressed the operations computed to get a prediction for any timestep from $t = 1$ to $t = \tau$, where c and b are bias vectors, and $tanh$ is a

non-linearity function.

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}, \tag{2.25a}$$

$$h^{(t)} = tanh(a^{(t)}), \tag{2.25b}$$

$$o^{(t)} = c + Vh^{(t)}, \tag{2.25c}$$

$$\hat{y}^{(t)} = softmax(o^{(t)}). \tag{2.25d}$$

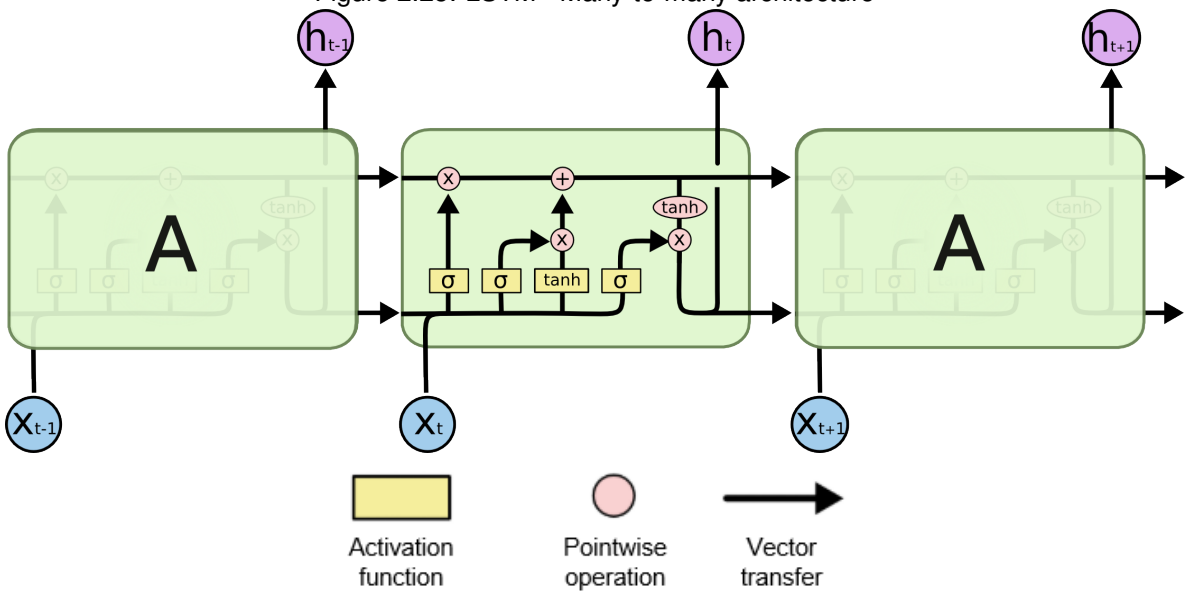
Furthermore, the Equations 2.25a and 2.25b are used at every timestep, in the many-to-one case, to be able to accumulate information until the end of the input sequence where the Equation 2.25c and 2.25d will be used. To compute this model's error with respect to its parameters W, V, U, b and c, a loss function must be used. Let's assume a loss function L, which has to be minimized with respect to each of the parameters mentioned. The gradient of the loss respect each of these has to be computed with backpropagation through time, and weights optimization will be found by using gradient descent. The backpropagation through time refers to the gradient that is computed through each timestep.

Long short-term memory (LSTM)

The long short-term memory architecture is a kind of RNN capable of learning long-term dependencies which its objective is to try to solve the vanishing and exploding gradient problem encounter with basic RNN. The vanishing gradient problem occurs when the gradient value of the loss with respect to a parameter becomes extremely small and does not contribute any more to the parameter update. The exploding gradient occurs when gradient value increases so much that the parameter update tends towards infinity. As in basic RNN, the gradient is computed through time, the probability of having vanishing and exploding gradient is increased by the number of timesteps to go through. Thus, having a basic RNN with long sequences as input may increase the probability for the model to forget what it learned at the beginning of a sequence and only allow it to have a short memory.

The LSTM works such as a basic RNN where it treats input data as sequence and also has a hidden layer A which receives a hidden state h, a cell state c and a component of the input sequence x at each timestep t, see Figure 2.25.

Figure 2.25: LSTM - Many-to-many architecture



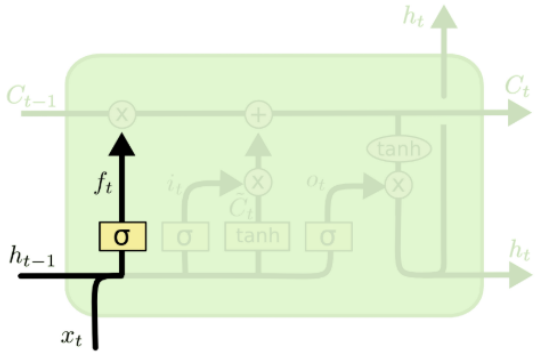
[41]

One of the differences with a basic RNN lies in the internal treatment of the hidden layer. Indeed, multiple sigmoid activation functions are applied to regulate the information that is passed through timesteps in the network and is commonly called gates. Here are the three sigmoid gates, also noted σ in Figure 2.25 :

- Forget gate,
- Input gate,
- Output gate.

When a component of x input sequence is given to the hidden layer of the LSTM, the forget gate is the first operation computed, see Figure 2.26 where h_{t-1} is the hidden state at the previous timestep, x_t is the current component of x and f_t is defined by Equation 2.26 where W_f is a weight matrix and b_f is a bias vector.

Figure 2.26: LSTM - Forget gate

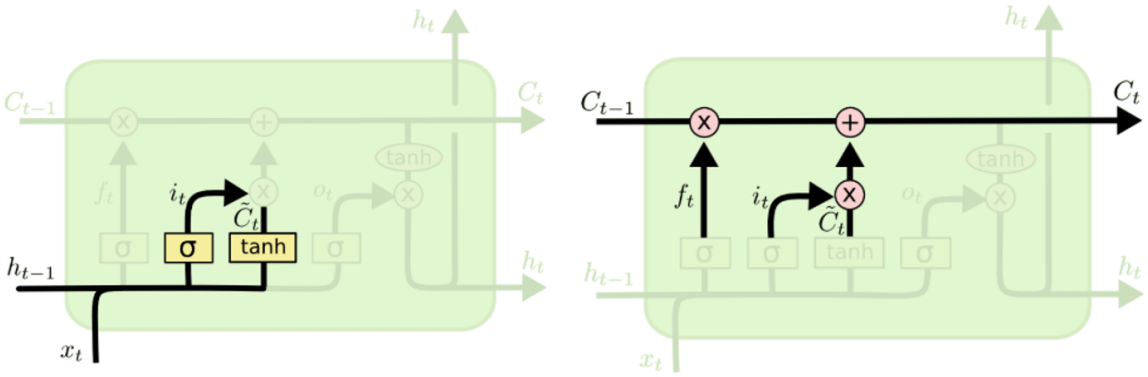


[41]

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f). \tag{2.26}$$

As a sigmoid function is applied on the previous information h_{t-1} and the current input component x_t which gives a value between 0 and 1, the role of the forget gate is to decide what information to throw away from the cell state c_{t-1} by a multiplication operation. Then, it is the turn of the input gate, see Figure 2.27 where $\tilde{C}_t$ is the new cell state and i_t, a vector used for scaling. $\tilde{C}_t$ and i_t are respectively defined by Equations 2.27a and 2.27b where W_i, W_c are weight matrix and b_i, b_c are bias vectors.

Figure 2.27: LSTM - Input gate



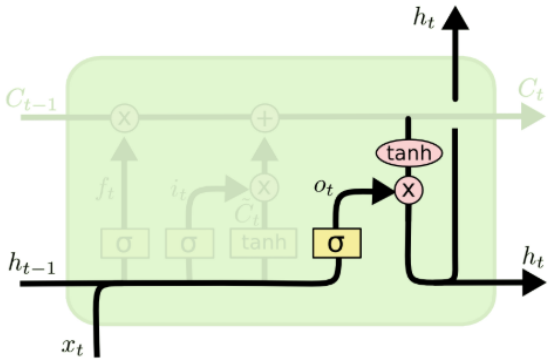
[41]

$$\tilde{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c), \tag{2.27a}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i). \tag{2.27b}$$

The role of the input gate is to decide what information to add to the cell state $\tilde{C_{t-1}}$ where i_t is a vector result of sigmoid application with values between 0 and 1 that will scale the update $\tilde{C}_t$ by a multiplication operation, see the left side of Figure 2.27. It will then add information, with a vector addition operation, to the cell state C_{t-1} that will be the final output cell state C_t of the current timestep. It is finally the turn of the output gate; see Figure 2.28 where h_t is the hidden state of the current timestep that is transmitted to the future timestep.

Figure 2.28: LSTM - Output gate



[41]

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \tag{2.28a}$$

$$h_t = o_t \cdot tanh(C_t). \tag{2.28b}$$

The role of the output gate is to decide what information to output to the future timestep. Through its sigmoid value between 0 and 1, the o_t component will scale the information to output by multiplication with C_t. A crucial property in LSTM is that the cell state only regulates the information to keep or remove in the hidden state, which is the information that navigates through timesteps and helps the model keep in memory the past input component information from x.

Convolutional neural network (CNN)

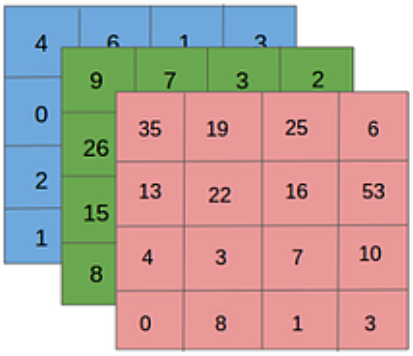
CNN is a feed-forward artificial neural network architecture that was biologically inspired on the working of the neurons of the visual cortex [43]. Hence, it has been used in many image classification

task [44], [45], but also in text classification [46], [47] and time series classification problem [48]–[50]. A typical CNN layer consists of three stages with the last layer that allows the network to make predictions [51]:

1. Convolution.
2. Non linearity.
3. Pooling.
4. Classification / Regression.

In order to discuss the fundamental operations of a CNN mentioned above, all the explanations will take an image classification task as an example. Moreover, an image can be considered a three-dimensional matrix, such that each dimension is an integer matrix, or channel, containing values between 0 and 255. Each of these dimensions refers to the separated Red, Green, and Blue (RGB) colors of the picture where each index contains the corresponding pixel color value, see Figure 2.29.

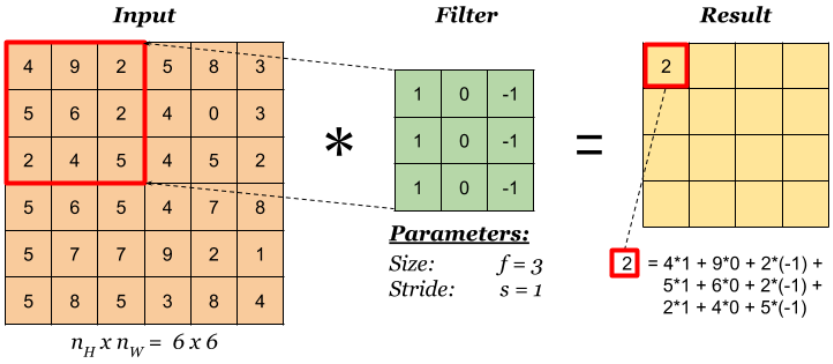
Figure 2.29: CNN - RGB image representation



[52]

The convolution layer aims to extract features from the input image while preserving the image pixels' spatial relationship. It is a method where an integer matrix, commonly called kernel or filter, is passed over an image and transform it based on filter values which can be mathematically formulated as an element-wise multiplication between a weight matrix W and sub-parts of the image x, see Figure 2.30 where x is an input of $n_H \cdot n_W$ size, filter size $f = 3 \cdot 3$, stride $s = 1$ and padding $p = 0$. The stride indicates the filter step on the image. It is essential to remember that filters act as a feature detector in the original input picture and give a result that can be called a feature map.

Figure 2.30: CNN - Convolution operation



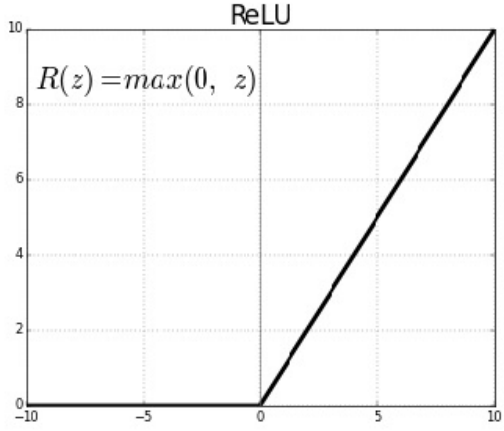
[53]

According to the Equation 2.29, the feature map F of size $m \cdot n$ is calculated such as the input image is denoted by x and the kernel filter by f. F rows and columns indexes are marked with m and n while the kernel filter matrix f rows and columns are noted as j and k, respectively.

$$F[m,n] = (x \cdot f)[m,n] = \sum_{j} \sum_{k} f[j,k]x[m-j,n-k]. \tag{2.29}$$

The next step is to introduce non-linearity in the network by applying an activation function, such as ReLU, on the convolution layer results. ReLU will be applied per pixel as an element-wise operation and will replace all negative values with zero; see Figure 2.31 where ReLU function $R(z)$ will take the maximum value between 0 and a given input z.

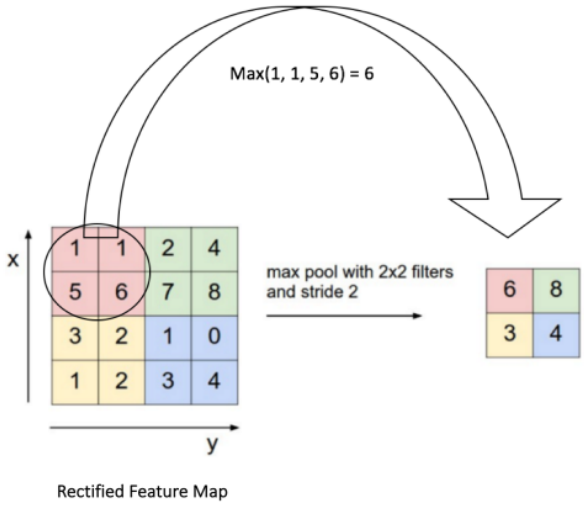
Figure 2.31: CNN - Non linearity function



[54]

Then, the spatial pooling has to be done in order to reduce the dimensionality of the feature map while keeping important information. The pooling can use different mathematical functions such as the maximum, the average, or the sum. In any case, the pooling operation helps to make the representation invariant to small translations of the input. Invariance to translation implies that if the input is translated by a small amount, the values of the pooled outputs do not change [51]. Assuming the max-pooling was chosen, a pooling size has to be set and by sliding the pooling window on the image, the maximum value in each region of the pooling window has to be taken, see Figure 2.32 where the stride $s = 2$ and the pooling window size is $2 \cdot 2$.

Figure 2.32: CNN - Max pooling



[55]

The three blocks discussed above (convolution, non-linearity, pooling) could be repeated to expand the network before having the last layer, which will be a fully connected one. This refers to a layer that flattens the last pooling layer into one dimension and maps it with an output layer. Indeed, the fully connected layer is an MLP with a final output layer that allows the model to make predictions. Such as the three layers that can be stacked multiple times to have a deeper network, it is common to use multiple fully connected layers with a progressive size reduction in the number of units before having the final output layer.

In addition to their ability to capture features in images, the CNN architecture also allows a computational cost reduction as each pooling layer reduces the size of the image initially given in input and, thus, reduces the total number of parameters to optimize in the network, compared to traditional artificial neural networks. In the different principal components example of CNN above, a convolution in two dimensions was used. However, one dimension convolution is also commonly used. Instead

of using the input data as a two-dimensional matrix, the one-dimensional convolution uses input as a vector. The operation is the same as two-dimensional convolution and is commonly used with time series data as they are usually represented in one dimension.

2.4.3 Regression

This section will discuss different model architectures used in this thesis for user privacy preservation. In order to use users mobile sensors data while not divulging personal data, some models would be considered in this section, such as generative models. This section's regression term refers to all its related models that will no try to solve a classification problem. Indeed, the models will try to map each input x with itself as x is composed of continuous time-series values that constitute, by definition, to a regression problem.

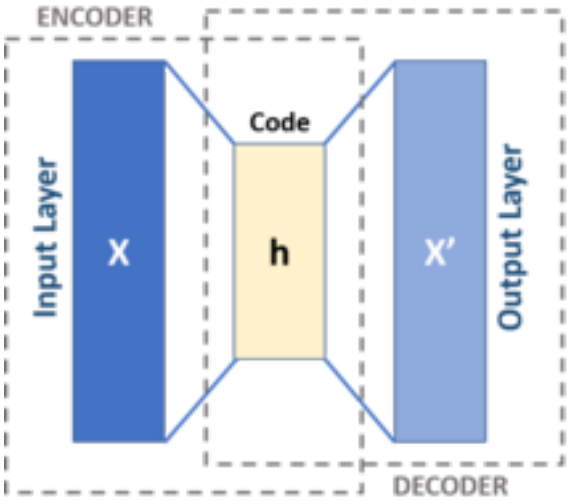
2.4.3.1 Generative models

Generative models are a branch of unsupervised learning, where unsupervised learning relates to models that work to discover patterns in the data and not predict a given label. Generative models can generate new samples that follow the same probabilistic distribution of a given dataset [56] by estimating the probability density function.

Auto encoder (AE)

The auto encoder (AE) is a type of artificial neural network that attempt to copy its inputs as outputs [42].The internal operation of the network may be viewed with two distinct parts, as illustrated in Figure 2.34 : the encoder function $f(x)$ which might be expressed as Equation 2.30a and the decoder function that try to produce a data reconstruction x' and formulated as Equation 2.30b.

Figure 2.33: Auto encoder



[57]

$$h = f(x), \tag{2.30a}$$

$$x' = g(h). \tag{2.30b}$$

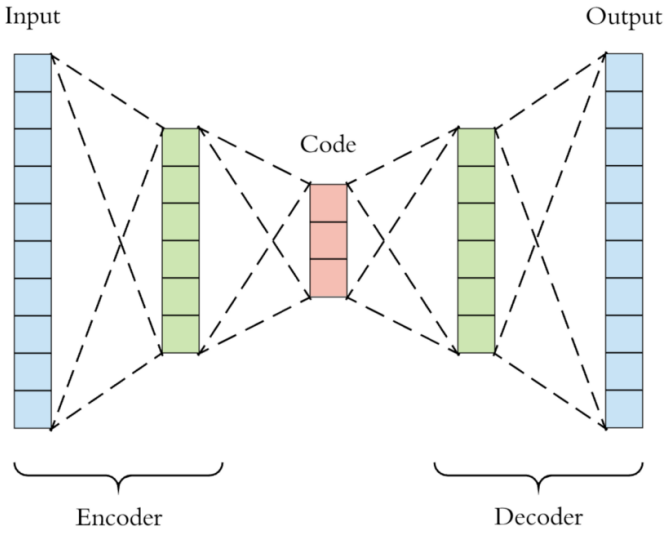
The AE objective is to reproduce the inputs as outputs. Thus, one may assume that the utility of such an architecture is that h, also called latent space or z, should ideally learn useful properties from x before reconstructing it as x'. Hence, the learning of useful properties might be more relevant than the decoder inputs reconstruction task itself. In order to obtain the most salient features of x, a standard method consists of constraining h of having fewer dimensions than x, and this architecture type is called an under complete AE [42]. The under complete AE with a linear decoder is commonly compared to the principal component analysis (PCA) as both learn a subspace of x and compute a dimensionality reduction as side-effect [58]. Moreover, h could have as many dimensions as x, and in the overcomplete case, it could have higher dimensions than x. In both cases, the decoder can learn to copy the inputs as outputs without necessarily learn useful h data distribution.

The loss function used in AE is usually the mean squared error where the objective is to minimize the dissimilarity of x' with x, see Equation 2.31 where θ is the parameter of f, φ is the parameter of g, and n is the number of instances in the data set.

$$L(\theta, \varphi) = \frac{1}{n} \sum_{i=1}^{n} (x_i - f_\theta(g_\varphi(x_i)))^2. \tag{2.31}$$

Concerning the AE architecture, both encoder and decoder could use any neural network architecture such as those described in section 2.4.2.3. However, the decoder must have the same number of layers as the reverse order's encoder part. That means, assuming an encoder with 3 layers going from an input size of 10 to an output size of 6, the decoder should have the same layers. However, the decoder inputs size will go from 6 to 10, in order to reduce dimensions first in the encoder and increase them in the decoder to recover the same shape as original inputs, see Figure 2.34.

Figure 2.34: Auto encoder - Layers disposition rule

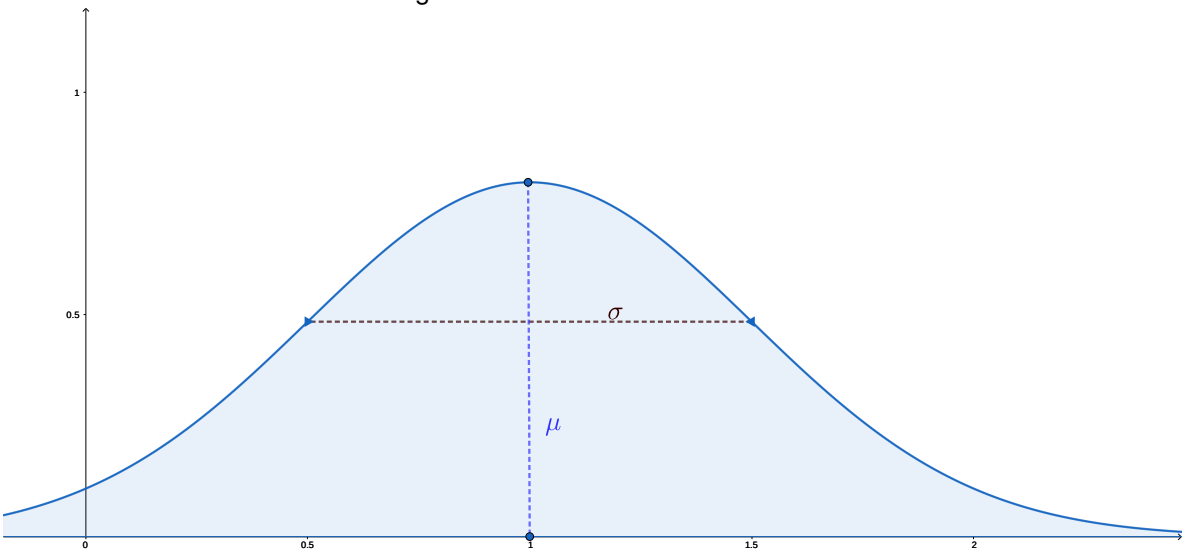


[59]

Variational auto encoder (VAE)

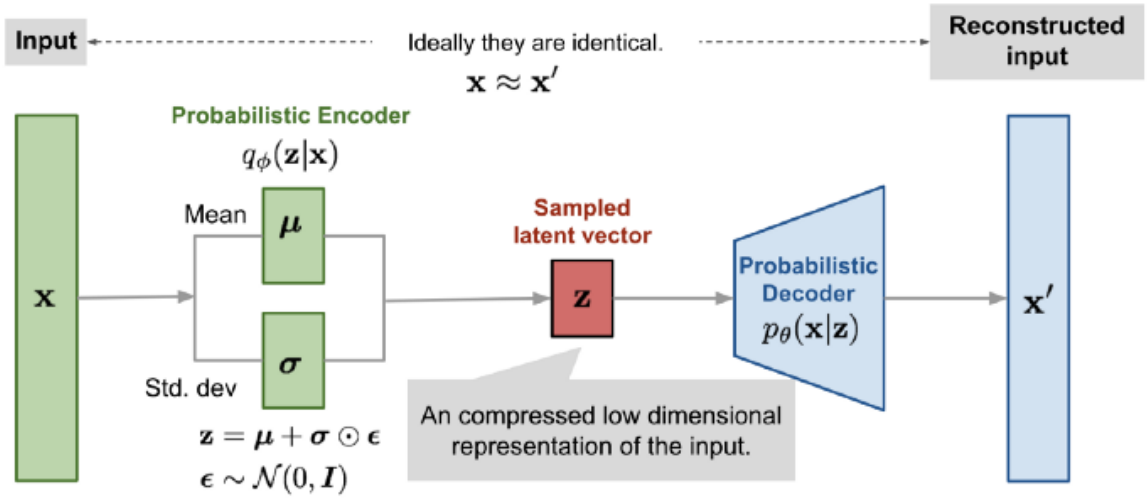
Variational autoencoder (VAE) is a regularized variant of autoencoders discussed in section 2.4.3.1 [42]. Its objective is to provide the ability to maximize the sparsity of the latent representation while limiting the difficulty of learning inputs properties in basic AE. Furthermore, VAE can generate new sample data from the latent space while it is not possible in AE. In order to generate new data following a specific Gaussian distribution, the mean and the standard deviation are needed. Indeed, the standard deviation gives a data dispersion measure, and the mean gives the center of the distribution. Assuming a Gaussian data distribution with a mean $\mu = 2$ and a standard deviation $\sigma = 0.5$, the data space could be illustrated as Figure 2.35.

Figure 2.35: Gaussian distribution



Given the above definition and such as VAE can generate a new data sample, a mean vector μ, and a standard deviation vector σ represent its latent distribution parameters. The VAE has the same purpose as AE that is to minimize the difference between inputs data x and reconstructed output x' from the decoder part while also having a latent representation vector noted z, see Figure 2.36 which assume a multivariate Gaussian.

Figure 2.36: VAE Gaussian structure



[60]

Some Bayes' theorem prerequisites must be defined as VAE is based on probability theory, see Equation 2.32 where $P(x)$ defines the probability of a random variable x to occur and $P(x|y)$ defines the probability of a random variable x to happen knowing that y has occurred. Bayes theorem also uses $P(y|x)$ as the posterior probability, $p(y)$ as the prior probability and Equation 2.32 as the likelihood ratio. Prior probability represents what is believed initially before new evidence is introduced and taken into account in posterior probability.

$$P(x|y) = \frac{P(x|y)P(y)}{P(x)}. \tag{2.32}$$

The Bayes theorem expressed above can be used with VAE that aims to find the best latent variable distribution which is also known as the posterior $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$, where z can be any distribution. The distribution family can be formulated as $q_\theta(z|x)$ which will be used to approximate $p(z|x)$. By using the Gaussian example of figure 2.36, θ refers to the parameters of the Gaussian distribution family that are the mean and the variance of the latent variables for each data point. Then, to measure the difference between the probability distribution $q_\theta(z|x)$ and $p(z|x)$, the Kullback-Leibler divergence (D_{KL}) has to be used which tells how well p is approximating, see equation 2.33.

$$D_{KL}(q_\theta(z|x)||p(z|x))) = E_q[log(q_\theta(z|x))] - E_q[log(p(x,z))] + log(p(x)). \tag{2.33}$$

[61]

Then, as minimizing the D_{KL} will optimize only the encoder part, another important thing has to be used that is the Evidence Lower Bound function. The VAE encoder and decoder parts are jointly trained by maximizing the Evidence Lower Bound (ELBO) that is formulated in Equation 2.34.

$$log(p_\theta(x|z))_{q_\theta(z|x)} - D_{KL}(q_\theta(z|x)||p(z|x)). \tag{2.34}$$

[61]

Where the first term is the reconstruction term, and the second term is the Kullback-Leibler divergence. The D_{KL} term can be interpreted as a regularizer that prevents the inference network from copying x into z [61]. Maximizing the ELBO function will increase the probability of observing data; thus, the negative ELBO is the VAE loss function to minimize.

2.5 Evaluation method

This section will discuss the evaluation methods used in the training and the testing phase of any models applied in this thesis. A cross-validation method was used where the data set discussed in

section 2.1 was divided into three subsets :

- Train,
- Validation,
- Test.

Each of those subsets is part of the global data set with a ratio of 70%, 15%, and 15%. The train set was used to train models to optimize their parameters while the test set was used to confirm the results on a trained model. In machine learning, it is common to use different data to train and test the model. The reason is that a model can overfit the data and learn to make satisfying results only on the training set and not be able to do the same on data that it has never seen. The validation set is only used with the artificial neural network models discussed in section 2.4.2.3, where the gradient descent has to be computed. It is used to validate the model during the training.

For information, all simple models of this work have been computed with the Sklearn python library's help, while artificial neural networks were done with the Pytorch library.

Classification

The metric used to evaluate each model performance, for the classification task, was the F1 score. F1 score is a combined metric between the precision and the recall where these 3 metrics use a confusion matrix between the correct labels to predict and the predicted ones of the model, as shown on Figure 2.37 where "Positive" and "Negative" may refer to the label 0 and 1 in a binary classification task. Each confusion matrix cell in Figure 2.37 is a combination between the model predictions and the original labels to predict.

Figure 2.37: Confusion matrix - True labels vs Predictions

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

[62]

The precision represents the proportion of positively predicted instances among all positive instances predicted by the model, while it can be formulated as Equation 2.35. In the carpooling classification

task of this work, this measure may be interpreted as the model capacity not to classify users as carpoolers when it is not the case.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive + False Positive}}. \tag{2.35}$$

On the other hand comes the recall metric, representing the proportion of right positively predicted instances among all positive instances in the entire data set, see Equation 2.36. This measure may be interpreted as the model capacity to classify carpoolers when it is the case.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}}. \tag{2.36}$$

Thus the f1 score metric is a combination of the precision and the recall discussed above; see Equation 2.37. This metric might be understood as a balance between precision and recall while according as much significance to True Positive as False Positive.

$$\text{F1 score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{\text{True Positive}}{\text{True Positive} + \frac{1}{2}(\text{False Positive} + \text{False Negative})}. \tag{2.37}$$

Regression

In the carpooling data encoding task, which is a regression problem, the root mean squared error (RMSE) was used. This measure is the rooted version of the average difference between the correct labels and the model predictions, called MSE, which was discussed in section 2.4.2.1. The RMSE is commonly used in the machine learning regression task as its results are more interpretable than MSE. Indeed, the MSE computes the squared average difference while RMSE applies a root on this result to provide RMSE results on the same scale as input data.

2.5.1 Training

During the training phase, all model hyperparameters needed to be optimized, where hyperparameters refer to the model parameters set before the training. Changing them could result in better model results; however, the best ones could depend on the complexity of the data and could be hard to find. A standard method is setting an interval of values for each hyperparameter and using the grid search algorithm. This method uses all combinations of hyperparameter values, while with n parameters and k possible values each, it will take k^n combinations. As this method may take some time, the Optuna optimization framework will be preferred in this thesis [63].

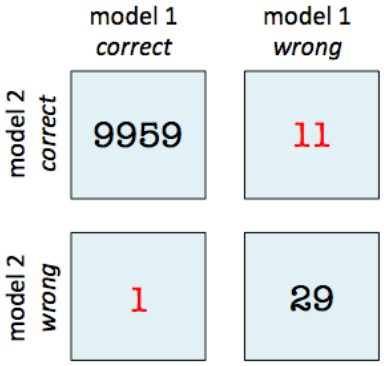
Optuna could be used with many optimization algorithms; however, only the Tree-structured Parzen Estimator (TPE) will be used and discussed in this thesis. TPE is a sequential model-based optimization (SMBO) strategy. SMBO methods successively construct models to estimate the performance of hyperparameters based on previous measurements; they will then choose new hyperparameters to be tested based on this model. The TPE strategy is based on the Bayesian probabilities theorem and uses the $P(y)$ and the $P(x|y)$, where y the performance measure of the model and x is the hyperparameters.

Moreover, in the case of artificial neural network models, the early stopping method has been applied. This method is used to stop the training of a model if a metric computed on the training set continue to converge but not the one computed on the validation set. It has to be done with an arbitrary number of epochs chosen, representing the number of times the model sees the entire training data set, where the model does not achieve to be better on the validation data set.

2.5.2 Testing

During the testing phase, the pre-trained models, where all hyperparameters were optimized in the training phase, must be tested on the test data that they have never seen. As the testing phase results could be related to the test data specific distribution, a commonly used technique in classification problems to ensure the significance of the results is the McNemar statistical hypothesis test. This statistical test is based on a contingency table between two classifiers, which compares the number of correct and wrong predictions made by two models, see Figure 2.38 where 9959 refers to the number of instances rightly predicted in both models, 11 refers to the number of wrong predictions made by model 1 that model 2 made correctly. A contingency table compares two models to each other rather than showing a single model's right and wrong predictions, such as in the confusion matrix shown in Figure 2.37.

Figure 2.38: McNemar - Contingency table example



[64]

The McNemar statistical test computed on a contingency table assumes a null hypothesis noted H_0 and a significance level noted δ. The null hypothesis assumes that two model results, even different, are equivalent. The objective is to apply the Chi-square formula, which is given by Equation 2.38, on the instances rightly classified by a model which are wrongly classified by the other, see the red box numbers in Figure 2.38. Then, the Chi-Square distribution table has to be used to map the Chi-square result with the p value interval given by Table [65]. If the p result of the Chi-square is lower than δ, H_0 is rejected, and the two model results are significantly different. With the values shown in the Figure 2.38 and a $\delta = 0.05$, the McNemar test will give a result of 8.33, see Equation 2.39, which corresponds to a p value interval of $0.002 < p < 0.005$ in the Chi-square table [65]. As p is lower than δ, the H_0 would be rejected, and the two model results would be considered significantly different.

$$\chi^2 = \frac{(b-c)^2}{b+c}, \tag{2.38}$$

$$\chi^2 = \frac{(M1_w - M2_w)^2}{M1_w + M2_w} = \frac{(11-1)^2}{11+1} \approx 8.33. \tag{2.39}$$

It is unlikely to use such a statistic test in a regression problem since there are no classes to predict but scalars instead. As discussed in section 2.5, the RMSE metric will evaluate the regression models used in this thesis either in the training phase than in the testing, where the only difference will be the data set employed.

3 Results

This section will present the different model architectures used in order to achieve the different tasks established in section 1.2. Moreover, as a reminder of section 2.1, each raw data row for the driver and the passenger represents a sensor frame capture with a collecting interval of 500ms. A general decision in this thesis was to prepare the data as windows of multiple frames, called $window\_size$, that slides over the data matrix by step of 1. The $window\_size$ tested values were chosen by following the Fibonacci sequence and starting from 5 to 21. The $window\_size$ could not have been any bigger as the GPU used for the models was an RTX 2060 Super with a limited cache memory of 8GB. Furthermore, based on the first model results, removing the gyroscope sensor from the data was decided. Its values did not vary pretty much throughout a trip, and some early models were using it as a dominant feature in their predictions. Thus, this section will only discuss results based on magnetometer and accelerometer sensor data without using the gyroscope.

3.1 Carpoolers classification - Original data

3.1.1 Data preparation

In this thesis, 5 model types have been chosen to be evaluated over the carpoolers classification problem, which is: logistic regression, SVM, MLP, RNN, LSTM, CNN. All of those models use the raw data discussed in section 2.1 with 2 different data preparation that can be grouped by model needs as follow :

1. Logistic regression, SVM, MLP, LSTM, RNN,
2. CNN.

The first data preparation cited above consists, for a given $window\_size$, to take the $window\_size$ frame rows of the driver, for example, and transpose them to columns, see Figure 3.1. In Figure 3.1, a $window\_size$ of 3 was chosen where each column of the upper matrix refers to a sensor data axis. The same process is performed on the passenger data matrix.

Then, driver and passenger matrices are merged for each travel where each column of one will be placed next to the corresponding column of the other, see Figure 3.3 where d refers to the driver and p to the passenger. The objective of this data preparation was to assign multiple frames to the same label and give more features to each model before asking it to make a prediction, as the data set will have $window\_size * k * n$ columns, instead of only k where k is the number of initial features, and n is the number of matrices merged.

Figure 3.1: Data preparation - Frames

Driver - Frames				
Acc_x	Acc_y	Acc_z	...	y
0.2	5	45	...	0
0.1	4	-1	...	0
0	2	-8	...	0
...	...	...	...	...

A matrix structure example of driver sensors data for a given travel

Figure 3.2: Data preparation - Windows

Driver - Frame windows							
Acc_x	Acc_x+1	Acc_x+2	Acc_y	Acc_y+1	Acc_y+2	...	y
0.2	0.1	0	5	4	2	...	0
...	...	...	...	...	...	...	...

Windows matrix data from Figure 3.1 which could be seen as the concatenation of each sensor axis of $window\_size$ *transposed*

Figure 3.3: Data preparation - Windows merged matrix

Driver, Passenger – Windows matrix					
Acc_x^d	Acc_x^p	Acc_x^d+1	Acc_x^p+1	Acc_x^d+2	Acc_x^p+2
0.2	18	0.1	4	0	122
...	...	...	...	...	...

Merged windows matrix between a passenger noted p*, and a driver noted* d

The second data preparation used for CNN aims to create windows from data frames as the first data preparation. The main difference with the first preparation lies in the resulting matrix where the three-axis x, y, and z will be separated and grouped by sensors, see Figure 3.4. As CNN is commonly known to be very good in image recognition and classification, the aim was to simulate a 3-dimensional space tensor in inputs as it would be the case for an image with the RGB color channels.

Figure 3.4: CNN data preparation

	Acc_x	Acc_x+1	Acc_x+2	Magne_x	Magne_x+1	Magne_x+2	y
W^{d1}	0.2	0.1	0	4	2	45	0
W^{p1}	...		Acc_y	Acc_y+1	Acc_y+2	Magne_y	Magne_y+1	Magne_y+2	y
W^{d2}	...	W^{d1}	0.2	0.1	0	4	2	45	0
W^{p2}	...	W^{p1}	...		Acc_z	Acc_z+1	Acc_z+2	Magne_z	Magne_z+1	Magne_z+2	y
...	...	W^{d2}	...	W^{d1}	0.2	0.1	0	4	2	45	0
W^{dn}	...	W^{p2}	...	W^{p1}	...	...	...	...	...	...	...
W^{pn}	8	...	...	W^{d2}	...	...	...	...	...	...	...
		W^{dn}	...	W^{p2}	...	...	...	...	...	...	...
		W^{pn}	8	...	...	...	...	...	...	...	...
				W^{dn}	...	...	...	...	...	...	...
				W^{pn}	8	0.33	0.58	1	2.6	3.8	0

Here, W refers to a window, y to the label,"Acc" is the abreviation of Accelerometer and "Magne" is the Magnetometer abreviation.

3.1.2 Model results

In this part, the results of the different models tested will be shown where each model's best parameters concerning the best F1 score achieved on the validation dataset have been kept, see Table 3.1 and 3.2. In linear models, 4 parameters have been trained, which are :

- C parameter,
- Gamma,
- Solver,
- Kernel.

The C parameter is a regularisation term used as an inverse term in the sklearn python library. It means that the C parameter is subtracted from 1 and as lower the C is, as stronger the regularization will be. The solver refers to the optimization algorithm used by sklearn, which is the lbfgs [66]. Then, the kernel algorithm and the gamma are only used with the SVM, which respectively specify the algorithm used to transform the data, see section 2.4.2.2 for more details, and the kernel coefficient.

In the artificial neural networks models, 5 hyperparameters have been trained, which are :

- Batch $\in [32, 512]$,
- Hidden size $\in [16, 64]$,
- Learning rate $\in [1e-10, 1e-2]$,
- Number of output channels $\in 1, 64$,
- Number of layers $\in [1, 3]$,
- Dropout $\in [0.1, 0.5]$.

The batch refers to the number of instances passed to the model before any weights parameters update. The learning rate is a coefficient applied on the weights update to control the optimization steps and is dynamically changed during the training with the Adam function [67], instead of using a fixed learning rate. In the CNN, the number of layers refers to the number of convolution, non-linearity, and

pooling stacked before the final dense layer. In the recurrent networks, the number of layers refers to the number of recurrent networks successively stacked, as recurrent networks have only one hidden layer. The hidden parameter is the number of units in each timestep in recurrent networks, while the output channels parameter is the number of feature map matrix given by a convolution step in CNN networks. Then, the dropout is a probability of units not being used in a given network layer. This parameter is commonly used to limit the overfitting problem and to allow the creation of deeper networks.

Concerning the results, the best linear model result is attributed to the logistic regression, which achieved an F1 score of 75.73% on test data while the LSTM in the neural networks category gets an F1 score of 93.76%. The best LSTM score is computed on a window size of 5, which means that the model has to see 2.5 seconds of sensors data to predict if two users were carpoolers. In order to better understand the F1 score given by the LSTM, it could be relevant to know its accuracy where it concerns how many True positive and True negative instances the model has predicted over the total number of instances that is 93.06%. The second-best model is the RNN with its 86.99% on the test data with also a window size of 5. Furthermore, The RNN achieved to have an accuracy of 87.74% and a recall of 85.03%.

Table 3.1: Linear models

	C param	Solver	Gamma	Kernel	Window size	F1 score
Logistic	1.05E-09	lbfgs	-	-	8	**75.73**%
SVM	5.80E-01	-	0.027	rbf	21	70.21%

Table 3.2: Neural network models

	Batch	Lr	Num layers	Hidden / Output chan.	Dropout	Window size	F1 score
CNN	160	1.17E-05	2	17	0.31	21	68.58%
LSTM	1664	0.0099	1	54	0	5	**93.76**%
RNN	1216	0.0049	1	36	0	5	86.99%

Then, to ensure the results of both LSTM and RNN are significant, which are the two models that have the best F1 score over the test data, the McNemar statistical test has to be done, as discussed in section 2.5. For this purpose, a significance level will be set to $\delta = 0.05$ with a null hypothesis noted H_0 that sustains the two model results are not significantly different. The Chi-square formula, expressed in Equation 2.38, has to be apply on the model result predictions shown on Table 3.3. If the given p value result is lower than the significance level, H_0 is rejected with RNN, and LSTM results considered significantly different. Then the χ^2 result formulated in Equation 3.1 is 543, with a freedom degree of 1 gives a p value of 5.74e-120. This p value has been found using the statsmodels python

library as the statistic McNemar value of 543 with a freedom degree of 1, can not be mapped with p in typical Chi-square tables. As a result, the LSTM and RNN classifier results are significantly different, where LSTM is the best.

Table 3.3: Rnn-Lstm - Contingency table

		LSTM	
		Correct	Wrong
RNN	Correct	15370	359
	Wrong	1312	885

Contingency table computed on test data where "Correct" refers to the number of instances rightly predicted and "Wrong" as the number of instances wrongly predicted

$$\chi^2 = \frac{(1312 - 359)^2}{1312 + 359} \approx 543$$
$$p = 5.74\text{e}{-120} \tag{3.1}$$

3.2 Data encoding

In this section, the different model architectures applied for the user data encoding task will be presented where the data preparation illustrated in Figure 3.3 has been employed. There are 2 types of encoding model that has been explored, which are the autoencoder and the variational autoencoder. The following 4 different architectures have experimented for the autoencoder:

- CNN,
- RNN,
- MLP,
- LSTM.

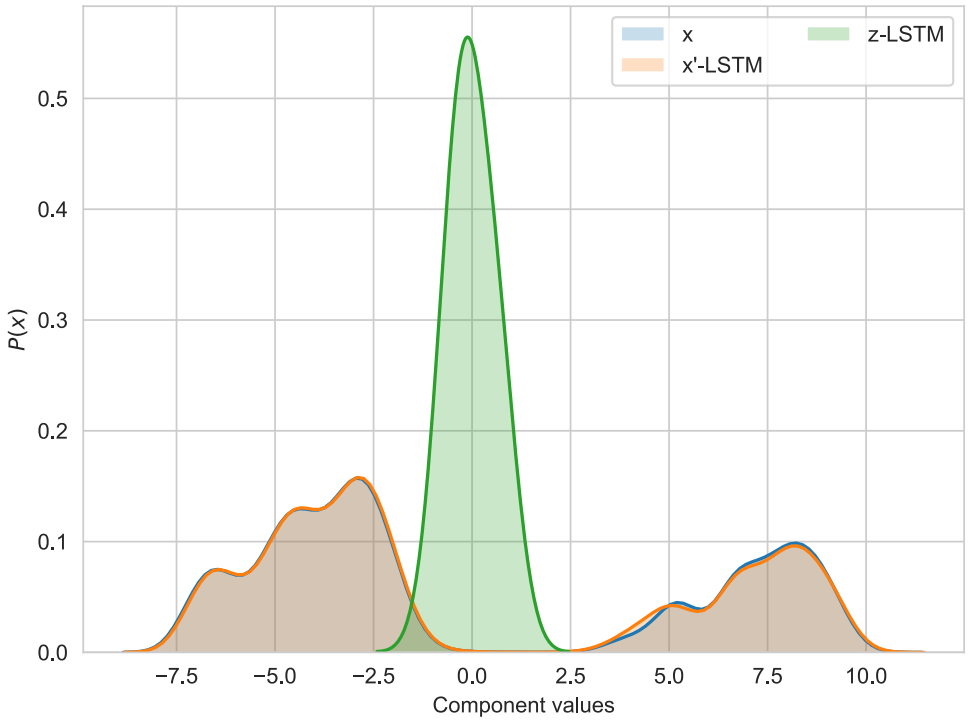
While only an MLP was explored for the variational autoencoder type due to the limited time at disposal, the results show that the minimum RMSE score is attributed to the LSTM model with a score of 0.92, see Figure 3.4. This result means that the test data passed in the network will have a mean reconstruction error of 0.92 at decoder output. To better understand whether this result is big or small, it could be compared to the mean and standard deviation on all data, which are respectively of 16.91 and 30.89. All hyperparameters expressed in Figure 3.4 have the same signification than in Figure 3.2, excepting the "z-dimensions". This hyperparameter refers to the latent state z dimensions between the encoder and the autoencoder model's decoder. Most of the autoencoder models have a z latent space size divided by 2 compared to the inputs so that the network can try to capture important inputs properties, see section 2.4.3.1 for more details.

Table 3.4: Autoencoder models

	Batch	Lr	Num layers	z-dimensions	Dropout	Window size	RMSE
MLP	192	0.0004	3	70	0.22	5	8.32
CNN	288	0.0055	2	17	0.14	5	30.04
LSTM	160	0.0018	1	35	0.04	5	**0.92**
RNN	448	0.0001	1	35	0.20	5	20.48
MLP-VAE	416	0.0031	1	17	0.25	5	40.27

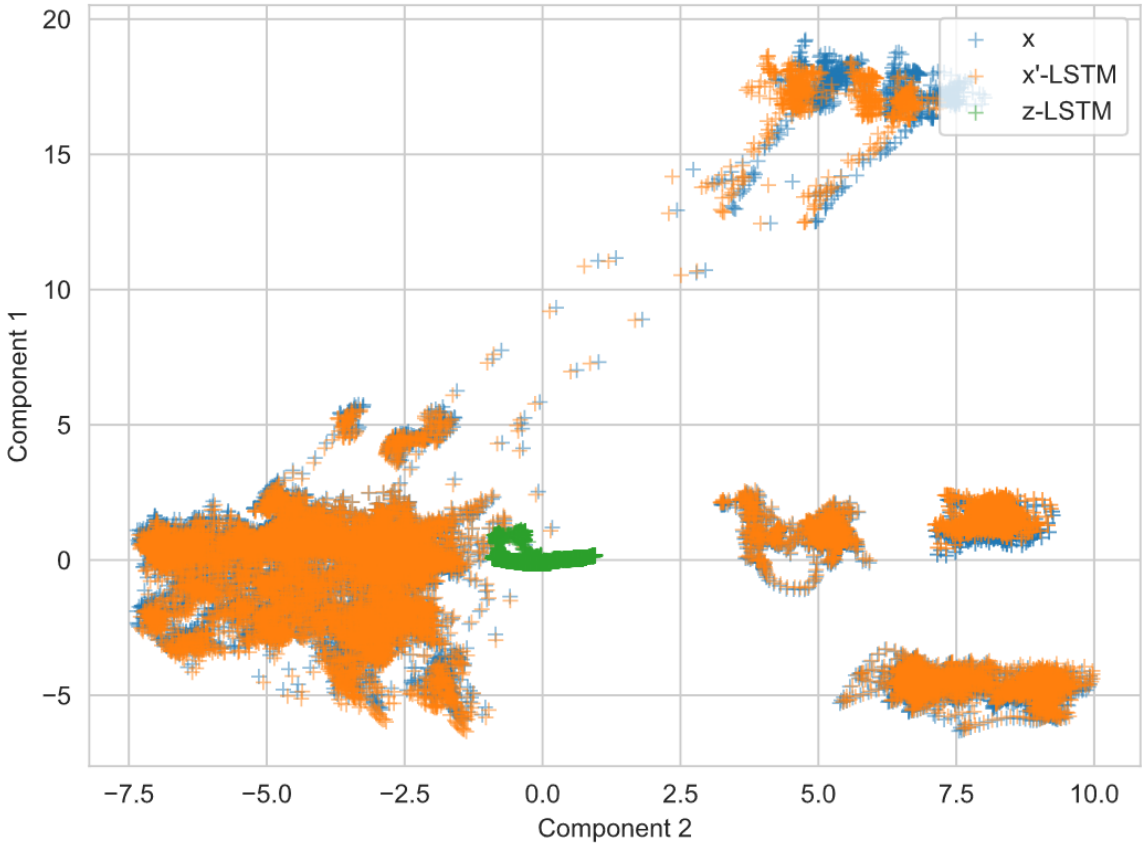
Furthermore, Figure 3.5 shows the comparison between the raw test data distribution noted x, the z test data distribution, and the decoded test data distribution noted x'. As a reminder of section 2.4.3.1, x is passed through the autoencoder, which encodes it to a z latent representation. Then the model tries a reconstruction, noted x', as near as possible to the initial input x. The Figure 3.5 shows that x and x' have almost same values while z seems to be pretty different with values oscillating between $[-2.5, 2.5]$. The Figure 3.6 is another way to see the data distribution similarity between x, z and x'. It uses the PCA to reduce distribution dimensionalities to 2 and allow to make a scatter plot with 2 components for each data distribution.

Figure 3.5: Probability density distribution with 1 component (w_size = 5)



PDF of test data noted x, x' and z given by the best LSTM autoencoder. Component values given by PCA with 1 component

Figure 3.6: Scatter plot of 2-components distributions (w_size = 5)



Scatter plot of test data noted x, x' and z given by the best LSTM autoencoder. Component values given by PCA with 2 component

3.3 Carpoolers classification - Encoded data

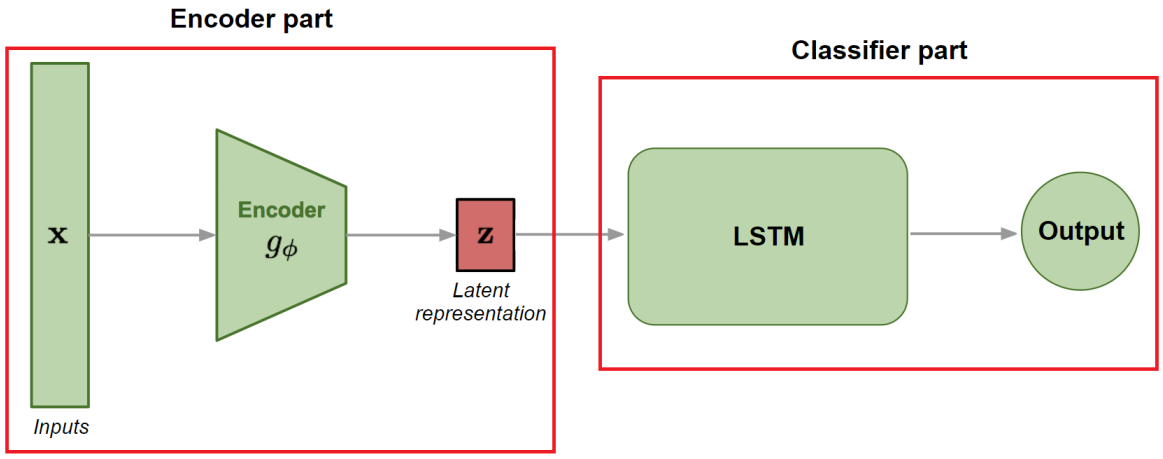
This section assumes that either z or x' could be used with the best carpoolers classifier, shown in section 3.1, without losing too much precision. Indeed, if the classifier achieved to have more than 93% of F1 score in its predictions, it could be relatively the same by using the LSTM decoded data x', as the data distribution is relatively similar, see Figure 3.5 and 3.6. Moreover, the x' reconstructed distribution is given by the LSTM autoencoder, which uses the latent z distribution; hence, z could contain the main properties of x and could also be used to classify carpoolers. Thus, the results shown in this section will use the pre-trained encoder part of the best autoencoder shown in Table 3.4, which is an LSTM. This autoencoder will transmit its z latent data representation to the best classifier shown in Table 3.2, which is also an LSTM. The complete structure of the given model is shown in Figure 3.7, where two training methods of this new architecture will be used :

1. The encoder part, shown in Figure 3.7, will have frozen weights and will not be trainable, as opposed to the classifier part.

2. Both the classifier and the encoder part, shown in Figure 3.7, will have trainable weights.

The first method objective is to preserve the significant difference between the z distribution and the input data x while classifying a carpooler trip correctly. The second approach is to optimize as much as possible all model weights to have the best predictions score while trying to preserve most of the distribution difference between z and x, which could not be guaranteed. The results presented in Table 3.5 refer to the best parameters obtained for the classifier part where the F1 score was computed on the test data set.

Figure 3.7: Classifier based on encoded data



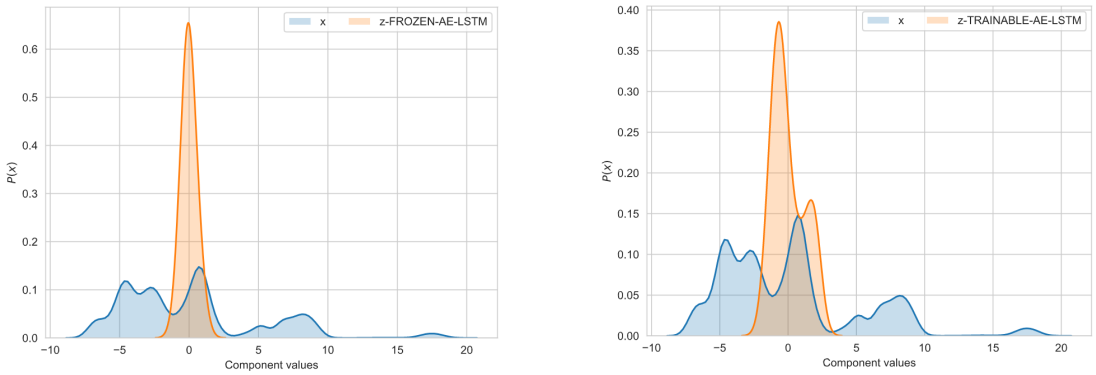
Architecture of the final classifier which receives the z latent representation, given by the encoder part of the best Autoencoder, as input.

Table 3.5: Classifier feeds by encoded data

	Batch	Lr	Num layers	z-dimensions	Dropout	Window size	F1 score
Frozen-AE-LSTM	320	0.0086	2	242	0.2	5	78.96%
Trainable-AE-LSTM	224	0.002	3	35	0.288	5	**87.47%**

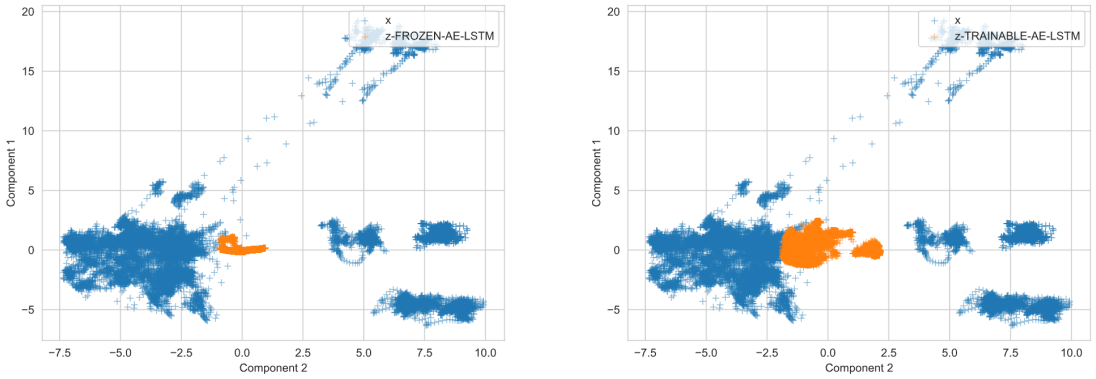
The "Frozen-AE-LSTM" model achieved an F1 score of 78.96%, which is not as great as the LSTM classifier results shown in Table 3.2, which used the raw data. The "Frozen" term refers to the fact that the encoder weights are initialized with the best autoencoder shown in Table 3.4 and are not trainable anymore. However, both classifier weights of the two models shown in Table 3.5 are trainable. Finally, Table 3.5 clearly shows that the LSTM autoencoder that is trainable stacked with the LSTM classifier that is also trainable gives the best results with an F1 score of 87.47%. Thus, it would be interesting to see how different the latent representation z is compared to the raw data for the two models discussed here to be sure that the trainable autoencoder always has a satisfying encoding rate comparing to its over-performing predictions results.

Figure 3.8: Probability density plot with 1 component between latent space z and raw data



Orange distributions noted z were generated by the encoder model. On the left, z is the result of the encoder with frozen weights while on the right the encoder has trainable weights

Figure 3.9: Scatter plot with 2 components between latent space z and raw data



Orange distributions noted z were generated by the encoder model. On the left, z is the result of the encoder with frozen weights while on the right the encoder has trainable weights

The left side of Figures 3.8 and 3.9 represents the raw data distribution in blue and the z latent space given by the "Frozen" encoder in orange while the "Trainable" model gives the right side of these Figures. The latent space is pretty well encoded in both models and does not seem to be over correlated to the raw distribution. Despite the z shown in Figures 3.8 and 3.9 seems to be a little less encoded in the "Trainable" encoder, its prediction F1 score is relatively good, being near the F1 score of 93.76% shown in Table 3.2 with the raw data.

4 Discussion

First of all, the results in Table 3.1, where the task was to classify windows of frames from a driver and a passenger in a trip, were obtained by using the robust linear classifiers SVM and the Logistic regression. However, these two classifiers do not perform as well as I expected. Indeed, the SVM is commonly known as a strong classifier that achieves to change the dimensional space before applying a linear classification between instances in a dataset that can have non-linear relations [68]. As shown in Figure 2.17 in section 2.4.2.2, the SVM seems to be capable of approaching complex data distributions. On the other hand, the logistic regression uses a non-linear logistic function on a linear regression on data that seems to be enough to capture data information and make great predictions. If we think about the data, it only represents user movement in a 3-dimensional space captured by the magnetometer and the accelerometer. Thus, if the axes between two users are aligned, a simple linear regression could be enough to make great predictions. In many recordings, it was intended not to align these dimensions; this could be why linear models such as SVM and the logistic regression did not perform as well as expected.

Secondly, CNN results shown in Table 3.2, which still be related to the classification task, were disappointing compared to the LSTM and RNN. Indeed, there is more than 25% of difference between the CNN and LSTM results, where even the logistic regression had better results. The first hypothesis was that the recurrent neural network type performs better on time series data [69]. However, this could not be the only reason which gives poor results as in CNN. The other reason could be the impossibility of creating a deeper network because of my GPU cache memory limited to 8GB and the time at disposal that was also limited. The solution would be to try to create deeper CNN models with more convolution layers to capture more feature properties in the data before entering the final dense layer that makes the predictions. With a limited GPU as mine, a solution could be to use smaller batches of data to use less memory and conduct a gradient update with fewer instances that could end up extending the training phase. Another possibility could be the data preparation made for CNN that could make data more difficult to understand for this model. Indeed, the data were prepared by separating each sensor axis in a different channel. Thus, this preparation could not let CNN apply filters among different axes and prevent them from capturing the relation between different sensor axes.

For the encoding task of user data, the results shown in Table 3.4 were pretty impressive for the LSTM architecture. As shown in Figure 3.5, the reconstructed distribution made by the LSTM was almost similar to the raw data, while the low dimensional latent data representation was pretty different. The other autoencoder results were pretty bad except for the MLP. The MLP achieved poor RMSE between raw data and reconstructed data, probably because its latent representation had as many

dimensions as the raw data. As discussed in section 2.4.3.1, this type of architecture could result in poor salient properties learning in the latent representation while having good reconstruction rate.

Moreover, the variation autoencoder gives a higher RMSE reconstruction rate with its MLP network. This lousy result could probably be due to the Kullback-Leibler divergence that uses the exponential variance to calculate the regularization part of the loss in a multivariate Gaussian assumption. As the variance is commonly estimated with a linear layer on the training dataset in practice, this could rapidly lead to an exploding divergence term. This kind of generative architecture should probably be applied with more attention to the first mean and variance estimation to have a stable learning curve to have better results. As a batch normalization layer has already been applied on data in the variational autoencoder before re-estimating the mean and the variance, a solution could be to ensure the Kullback-Leibler divergence term is applied on data values clipped between 0 and 1.

Finally, the classification of carpoolers using encoded windows of sensor data gives some great results, see Table 3.5. Instead of using raw data to accomplish the classification task, we used the best autoencoder shown in Table 3.4 to get latent representations of data to feed the best classifier model architecture. The first approach was to use the pre-trained autoencoder that had minimized its reconstruction error, resulting in the most salient features of data in the latent representation. Using this model, we wanted to have the best representation of inputs to give to the classifier and train it with these types of data. Unfortunately, the classifiers trained with latent representations, without the possibility to train a bit more the weights in the autoencoder, give results about 78% of f1 score. The second approach was also to use the best pre-trained autoencoder to generate latent representations of data and feed the classifier with them. However, we assumed that the pre-trained autoencoder was a form of weights initialization where the full model task will be to maximize its correct predictions while minimizing the distribution resemblance between raw data passed in autoencoder and underlying representations. The second approach worked better by giving results around 87% with around 6% less than models feed by raw data. A third approach could have been to use both encoder and decoder parts of the autoencoder, where each would be on a different machine that could simulate two distant hosts. The encoder would receive raw data that it transmits to the decoder while the classifier would be fed by decoded data that are almost the same as the raw data, see Figure 3.5. This approach could have given results as good as raw data while keeping user data encoded between the encoder and the decoder host. However, with the usage of latent representations of data, nobody can read a data distribution that is near the raw data, and this leads to better privacy data protection.

4.1 Limitations

This section will talk about the limitations of this work and the reproducibility of the results presented. This research thesis was made in a controlled environment, especially in the data collection, and could have better results than real-world situations. Indeed, the data was collected by a mobile application that was used mainly by the same person with multiple smartphones. Two primary use cases have been used to collect the data. The first one was to carry a smartphone in the pocket and another one on a nearby seat to simulate two people in the same car. The second one was to keep a smartphone in one car and put another one in another car and then drive in two different directions. Those aim to capture opposed cases to give data to models that have a less ambiguous separation and more variation in data. Thus, the dataset used in this thesis may contain some biases related to the fact that the data was generated using only a few people driving style on a few road trips.

Moreover, this work aimed to test models on data collected in a controlled environment to see their performance to go through a second phase where the application would be given to Mobilidée employees to collect data closer to the real carpoolers usage.

Another limitation was the material at disposal. All models were trained and tested on a machine, which has a Ryzen 5 3700x as CPU, 32 GB of RAM, and an RTX 2060 Super with 8GB cache memory as GPU. This setup was sufficient to run all models discussed in this work; however, it might be necessary to have better setup or to compute models on cloud solutions to explore deeper models. This limitation is linked with the time to do this work. More the hardware used is powerful to run machine learning models, the less time is consumed to complete the training and testing phase. So, having a more powerful machine could have been lead to more exploration in models discussed.

5 Conclusion

In this work, several machine learning algorithms were used to try to assess carpooling using low-invasive mobile sensors. The first objective was mainly to classify carpool trips, where some people traveled together, and others did not. The second objective was to be able to ensure user privacy while analyzing their data.

Since no data set already existed at the time of this thesis and to be able to classify carpool trips, we had to create an application to collect data from different sensors, such as the accelerometer, the magnetometer, and the gyroscope.

Once the data were collected, the first part of machine learning models application was done with some linear models such as SVM and logistic regression and artificial neural networks such as LSTM, RNN, CNN, and MLP. The LSTM architecture was the one who achieves the best result, with more than 93% of the f1 score over the carpoolers classification task. The second part of machine learning was then done with autoencoder models using MLP, RNN, LSTM, and CNN networks, while the best model was also the one using the LSTM with an RMSE reconstruction error around 0.92. The final machine learning application was made using the best autoencoder and the best classifier architectures to encode user sensor data while ensuring satisfying predictions in the classification task where an f1 score of around 87% was obtained.

For future works, various other exploration techniques might be used, such as the Euclidean distance between a driver's accelerometer and a passenger that was previously aligned using the gyroscope. Each smartphone sensor's axis has a fixed position relative to the smartphone position, which makes it challenging to analyze sensors from two smartphones in different positions. Thus, the sensor fusion approach could be used where it combines the information from the accelerometer and the gyroscope to find an optimal calibration using the maximum likelihood [70].

Another perspective for future work could be to use the sensor fusion approach discussed above and try to use the resultant values to feed some machine learning models, such as those explored in this thesis. Another solution could be to use the PCA algorithm on each 3-dimensional mobile sensor data to reduce the dimensionality space to 1 dimension by sensors and try to use it with machine learning models.

Bibliography

[1] J. Jung and Y. Koo, “Analyzing the Effects of Car Sharing Services on the Reduction of Greenhouse Gas (GHG) Emissions”, *Sustainability*, vol. 10, no. 2, p. 539, Feb. 2018. DOI: 10.3390/su10020539. [Online]. Available: https://www.mdpi.com/2071-1050/10/2/539 (visited on 02/04/2020).

[2] Blablacar.fr. (2020). Bus ou covoiturage ? Voyagez moins cher, [Online]. Available: https://www.blablacar.fr/ (visited on 02/04/2020).

[3] Waze.com. (2020). Couvrez vos frais en covoiturant avec Waze Carpool, [Online]. Available: https://www.waze.com/fr/carpool (visited on 02/04/2020).

[4] mobilidee.ch. (2020). Gestion, [Online]. Available: https://mobilidee.ch/gestion/ (visited on 02/04/2020).

[5] eur-lex.europa.eu. (2020). EUR-Lex - 32016R0679 - EN - EUR-Lex, [Online]. Available: https://eur-lex.europa.eu/legal-content/FR/ALL/?uri=CELEX%3A32016R0679 (visited on 02/04/2020).

[6] M. Malekzadeh, R. G. Clegg, A. Cavallaro, and H. Haddadi, “Mobile Sensor Data Anonymization”, *Proceedings of the International Conference on Internet of Things Design and Implementation - IoTDI ’19*, pp. 49–58, 2019. DOI: 10.1145/3302505.3310068. arXiv: 1810.11546. [Online]. Available: http://arxiv.org/abs/1810.11546 (visited on 11/01/2019).

[7] M. Malekzadeh, R. G.Clegg, A. Cavallaro, and H. Haddadi, “Privacy and utility preserving sensor-data transformations”, *Pervasive and Mobile Computing*, vol. 63, p. 101 132, Nov. 2, 2019, ISSN: 1574-1192. DOI: 10.1016/j.pmcj.2020.101132. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574119220300201.

[8] S. C. Maina, R. E. Bryant, W. O. Ogallo, K. R. Varshney, S. Speakman, C. Cintas, A. Walcott-Bryant, R. Samoilescu, and K. Weldemariam, “Preservation of anomalous subgroups on variational autoencoder transformed data”, in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 3627–3631. DOI: 10.1109/ICASSP40776.2020.9054495.

[9] N. C. Abay, Y. Zhou, M. Kantarcioglu, B. Thuraisingham, and L. Sweeney, “Privacy preserving synthetic data release using deep learning”, in *Machine Learning and Knowledge Discovery in Databases*, M. Berlingerio, F. Bonchi, T. Gärtner, N. Hurley, and G. Ifrim, Eds., Cham: Springer International Publishing, 2019, pp. 510–526, ISBN: 978-3-030-10925-7.

[10] N.-D. T. Tieu, H. H. Nguyen, H.-Q. Nguyen-Son, J. Yamagishi, and I. Echizen, “An approach for gait anonymization using deep learning”, in *2017 IEEE Workshop on Information Forensics and Security (WIFS)*, Dec. 2017, pp. 1–6. DOI: 10.1109/WIFS.2017.8267657.

[11] A. Abdrashitov and A. Spivak, “Sensor data anonymization based on genetic algorithm clustering with L-Diversity”, in *2016 18th Conference of Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT)*, Apr. 2016, pp. 3–8. DOI: 10.1109/FRUCT-ISPIT.2016.7561500.

[12] S. Kuk, Y. Park, S. Han, and H. Kim, “Car-level co-location detection on trains for infectious disease epidemic monitoring”, *Electronics Letters*, vol. 53, no. 1, pp. 8–10, 2017, ISSN: 0013-5194. DOI: 10.1049/el.2016.3295.

[13] K. A. Nguyen, C. Watkins, and Z. Luo, “Co-location epidemic tracking on London public transports using low power mobile magnetometer”, in *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Sep. 2017, pp. 1–8. DOI: 10.1109/IPIN.2017.8115963.

[14] J. Lester, B. Hannaford, and G. Borriello, “
Are You with Me?” – Using Accelerometers to Determine If Two Devices Are Carried by the Same Person”, in *Pervasive Computing*, A. Ferscha and F. Mattern, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2004, pp. 33–50, ISBN: 978-3-540-24646-6. DOI: 10.1007/978-3-540-24646-6_3.

[15] M. Zeng, L. T. Nguyen, B. Yu, O. J. Mengshoel, J. Zhu, P. Wu, and J. Zhang, “Convolutional Neural Networks for human activity recognition using mobile sensors”, in *6th International Conference on Mobile Computing, Applications and Services*, Nov. 2014, pp. 197–205. DOI: 10.4108/icst.mobicase.2014.257786.

[16] E. Nemati, C. Batteate, and M. Jerrett, “Opportunistic Environmental Sensing with Smartphones: A Critical Review of Current Literature and Applications”, *Current Environmental Health Reports*, vol. 4, no. 3, pp. 306–318, Sep. 2017, ISSN: 2196-5412. DOI: 10.1007/s40572-017-0158-8. pmid: 28879432.

[17] S. Subhanjan, C. Samrat, G. Amit Kr., B. Indrajit, and M. Tamal. (2015). TrackMe - a low power location tracking system using smart phone sensors - IEEE Conference Publication, [Online]. Available: https://ieeexplore.ieee.org/document/7411226 (visited on 11/28/2019).

[18] S. Panichpapiboon and P. Leakkaw, “Traffic Sensing Through Accelerometers”, *IEEE Transactions on Vehicular Technology*, vol. 65, no. 5, pp. 3559–3567, May 2016, ISSN: 0018-9545, 1939-9359. DOI: 10.1109/TVT.2015.2448237. [Online]. Available: http://ieeexplore.ieee.org/document/7130669/ (visited on 11/15/2019).

[19] I. M. G. Hajar, I. H. Muhammad, A. Z. Zuhri, and N. Siti, "MotoSOS: Accident Detection for Motorcycle Riders Using Motion Sensors", *Journal of Advanced Research in Computing and Applications*, 2019. [Online]. Available: http://www.akademiabaru.com/doc/ARCAV15_N1_P9_19.pdf (visited on 07/24/2020).

[20] Android. (May 26, 2020). Mobile sensor's types, [Online]. Available: https://source.android.com/devices/sensors/sensor-types?hl=fr (visited on 07/24/2020).

[21] R. Goodrich. (2013). Accelerometers: What They Are and How They Work, [Online]. Available: https://www.livescience.com/40102-accelerometers.html (visited on 07/24/2020).

[22] A. Huang, S. Gao, Y. Dai, V. Kitsos, W. Tian, and L. Xu, "A Capacitive Information-Based Force-Voltage Responsivity Stabilization Method for Piezoelectric Touch Panels", *IEEE Journal of the Electron Devices Society*, vol. 7, pp. 1018–1025, 2019, ISSN: 2168-6734. DOI: 10.1109/JEDS.2019.2939912.

[23] w3.org. (May 3, 2019). Magnetometer, how it works and security issues, [Online]. Available: https://www.w3.org/TR/magnetometer/ (visited on 07/24/2020).

[24] techahead. (Jul. 24, 2020). How does a Gyroscope sensor work in your smartphone?, [Online]. Available: https://www.techaheadcorp.com/knowledge-center/how-gyroscope-sensor-work-in-smartphone/ (visited on 07/24/2020).

[25] Earth.esa.int. (Jul. 24, 2020). Gravity in detail - Content - Earth Online - ESA, [Online]. Available: https://earth.esa.int/web/guest/-/gravity-in-detail-5728 (visited on 07/24/2020).

[26] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data", *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, Jun. 1, 2004, ISSN: 1931-0145. DOI: 10.1145/1007730.1007735. [Online]. Available: https://doi.org/10.1145/1007730.1007735 (visited on 07/26/2020).

[27] A. Dotis. (May 3, 2019). Autocorrelation in Time Series, [Online]. Available: https://medium.com/@dganais/autocorrelation-in-time-series-c870e87e8a65 (visited on 07/26/2020).

[28] math-stat.unibe.ch, "Time Series", 2015. [Online]. Available: https://www.math-stat.unibe.ch/e237483/e237655/e243381/e281679/files281691/Chap12_ger.pdf (visited on 08/05/2020).

[29] mathinsight.org. (2020). Magnitude of a vector definition - Math Insight, [Online]. Available: https://mathinsight.org/definition/magnitude_vector (visited on 07/26/2020).

[30] R. S. Bhat. (Jan. 14, 2020). Why not MSE as a loss function for logistic regression?, [Online]. Available: https://towardsdatascience.com/why-not-mse-as-a-loss-function-for-logistic-regression-589816b5e03c (visited on 08/01/2020).

[31] datacademia.com. (2020). Machine Learning - (Univariate|Simple) Logistic regression, [Online]. Available: https://datacadamia.com/data_mining/simple_logistic_regression (visited on 07/31/2020).

[32] M. Kuhn and K. Johnson. (2016). Applied predictive modeling, [Online]. Available: https://link.springer.com/content/pdf/10.1007%2F978-1-4614-6849-3.pdf (visited on 07/31/2020).

[33] V. Vladimir. (1995). Support-Vector Networks, [Online]. Available: http://image.diku.dk/imagecanon/material/cortes_vapnik95.pdf (visited on 08/01/2020).

[34] javatpoint.com. (2020). Support Vector Machine (SVM) Algorithm - Javatpoint, [Online]. Available: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm (visited on 08/01/2020).

[35] J.-p. Vert, K. Tsuda, and B. Schölkopf. (2004). A primer on kernel methods, [Online]. Available: http://members.cbio.mines-paristech.fr/~jvert/publi/04kmcbbook/kernelprimer.pdf (visited on 08/01/2020).

[36] F. Rosenblatt, *The Perceptron: A Probabilistic Model for Information Storage and Organization*. 1958.

[37] M. Deshp and e. (Sep. 12, 2017). Perceptrons: The First Neural Networks, [Online]. Available: https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/ (visited on 08/02/2020).

[38] Z. Ali, I. Hussain, M. Faisal, H. M. Nazir, T. Hussain, M. Y. Shad, A. Mohamd Shoukry, and S. Hussain Gani. (May 2, 2017). Forecasting Drought Using Multilayer Perceptron Artificial Neural Network Model, [Online]. Available: https://www.hindawi.com/journals/amete/2017/5681308/ (visited on 08/02/2020).

[39] M. A. Nielsen, "Neural Networks and Deep Learning", 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com (visited on 08/02/2020).

[40] M. Skorski, "Chain Rules for Hessian and Higher Derivatives Made Easy by Tensor Calculus", Nov. 29, 2019. arXiv: 1911.13292 [cs]. [Online]. Available: http://arxiv.org/abs/1911.13292 (visited on 08/05/2020).

[41] Colah. (2015). Understanding LSTM Networks – colah's blog, [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 08/04/2020).

[42] I. Goodfellow, Y. Bengio, and A. Courville. (2019). MIT Deep Learning book, [Online]. Available: https://www.deeplearningbook.org/ (visited on 08/04/2020).

[43] K. Fukushima. (1980). Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position, [Online]. Available: https://link.springer.com/content/pdf/10.1007/BF00344251.pdf (visited on 08/05/2020).

[44] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu. (2016). CNN-RNN: A Unified Framework for Multi-label Image Classification, [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Wang_CNN-RNN_A_Unified_CVPR_2016_paper.pdf (visited on 08/05/2020).

[45] S. Yu, S. Jia, and C. Xu. (2017). Convolutional neural networks for hyperspectral image classification, [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0925231216310104 (visited on 08/05/2020).

[46] J. Liu, Y. Wu, W.-C. Chang, and Y. Yang. (2017). Deep Learning for Extreme Multi-label Text Classification, [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3077136.3080834 (visited on 08/05/2020).

[47] F. Dernoncourt, "Sequential short-text classification with neural networks", Thesis, Massachusetts Institute of Technology, 2017. [Online]. Available: https://dspace.mit.edu/handle/1721.1/111880 (visited on 09/27/2020).

[48] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline", in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 1578–1585. DOI: 10.1109/IJCNN.2017.7966039.

[49] S. Ha and S. Choi, "Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors", in *2016 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2016, pp. 381–388. DOI: 10.1109/IJCNN.2016.7727224.

[50] J. B. Yang, M. N. Nguyen, P. P. San, X. L. Li, and S. Krishnaswamy. (2015). Deep Convolutional Neural Networks On Multichannel Time Series For Human Activity Recognition, [Online]. Available: https://www.aaai.org/ocs/index.php/IJCAI/IJCAI15/paper/view/10710/11297 (visited on 08/05/2020).

[51] I. Goodfellow, Y. Bengio, and A. Courville, *Adaptive Computation and Machine Learning Series-Deep Learning-The MIT Press (2016)*. 2016, ISBN: 978-0-262-03561-3. [Online]. Available: https://www.academia.edu/38223830/Adaptive_Computation_and_Machine_Learning_series_Deep_learning_The_MIT_Press_2016_pdf (visited on 08/05/2020).

[52] thelearningmachine.ai. (2020). TLM | Convolutional Neural Network (CNN), [Online]. Available: https://www.thelearningmachine.ai/cnn (visited on 08/05/2020).

[53] B. Pembelajaran. (Mar. 7, 2018). Student Notes: Convolutional Neural Networks (CNN) Introduction, [Online]. Available: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/ (visited on 08/05/2020).

[54] S. Bhattarai. (2018). Relu-activation-function, [Online]. Available: https://saugatbhattarai.com.np/what-is-activation-functions-in-neural-network-nn/relu-activation-function/ (visited on 08/05/2020).

[55] ujjwalkarn. (Aug. 10, 2016). An Intuitive Explanation of Convolutional Neural Networks, [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ (visited on 08/05/2020).

[56] cs.toronto.edu. (2017). Deep Generative Models, [Online]. Available: https://www.cs.toronto.edu/~slwang/generative_model.pdf (visited on 08/06/2020).

[57] Wikipedia, *Autoencoder*, in *Wikipedia*, Jul. 14, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Autoencoder&oldid=967729710 (visited on 08/06/2020).

[58] G. Hinton and R. Salakhutdinov. (2006). Reducing the Dimensionality of Data with Neural Networks, [Online]. Available: https://science.sciencemag.org/content/sci/313/5786/504.full.pdf (visited on 08/06/2020).

[59] M. S. Researcher PhD. (). Comprehensive Introduction to Autoencoders, [Online]. Available: https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368 (visited on 08/06/2020).

[60] L. Weng. (Aug. 12, 2018). From Autoencoder to Beta-VAE, [Online]. Available: https://lilianweng.github.io/2018/08/12/from-autoencoder-to-beta-vae.html (visited on 08/06/2020).

[61] L. Chen, S. Dai, Y. Pu, E. Zhou, C. Li, Q. Su, C. Chen, and L. Carin, "Symmetric variational autoencoder and connections to adversarial learning", A. Storkey and F. Perez-Cruz, Eds., ser. Proceedings of Machine Learning Research, vol. 84, Playa Blanca, Lanzarote, Canary Islands: PMLR, Apr. 9–11, 2018, pp. 661–669. [Online]. Available: http://proceedings.mlr.press/v84/chen18b.html (visited on 09/27/2020).

[62] K. P. Shung. (). Accuracy, Precision, Recall or F1?, [Online]. Available: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9 (visited on 08/09/2020).

[63] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework, [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3292500.3330701 (visited on 08/10/2020).

[64] S. Raschka. (2020). Contigency Table for McNemar's Test - mlxtend, [Online]. Available: http://rasbt.github.io/mlxtend/user_guide/evaluate/mcnemar_table/ (visited on 08/11/2020).

[65] Zach. (Sep. 18, 2018). Chi-square Distribution Table, [Online]. Available: https://www.statology.org/chi-square-distribution-table/ (visited on 08/11/2020).

[66] J. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. Sagastizabal, *Numerical Optimization – Theoretical and Practical Aspects*. Jan. 1, 2006.

[67] A. D.Bull, "Convergence Rates of Efficient Global Optimization Algorithms", *Journal of Machine Learning Research*, 2011. [Online]. Available: https://www.jmlr.org/papers/volume12/bull11a/bull11a.pdf (visited on 09/27/2020).

[68] M. Martinez-Ramon, J. L. Rojo-Alvarez, G. Camps-Valls, J. Munoz-Mari, n. Navia-Vazquez, E. Soria-Olivas, and A. R. Figueiras-Vidal, "Support Vector Machines for Nonlinear Kernel ARMA System Identification", *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1617–1622, Nov. 2006, ISSN: 1941-0093. DOI: 10.1109/TNN.2006.879767.

[69] J. Connor, R. Martin, and L. Atlas, "Recurrent neural networks and robust time series prediction", *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 240–254, Mar. 1994, ISSN: 1941-0093. DOI: 10.1109/72.279188.

[70] F. Olsson, M. Kok, K. Halvorsen, and T. B. Schön, "Accelerometer calibration using sensor fusion with a gyroscope", in *2016 IEEE Statistical Signal Processing Workshop (SSP)*, Jun. 2016, pp. 1–5. DOI: 10.1109/SSP.2016.7551836.