UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Master Degree in Computer Science

Data analytics for vehicular traffic networks: a case study

ADVISOR:
Prof. Alberto Ceselli

CO-ADVISOR:
Dott. Andrea Tironi

CANDIDATE:
Matteo Gandossi
921173

Academic Year 2018/2019

Contents

Chapter 1

Introduction

Traffic control is a major source of concerns at different levels, introducing challenges which range from pure logistics management to societal issues like pollution and crime control.

That is why information systems to monitor vehicular traffic are gaining more and more relevance. It is the case, for instance, of the region around the city of Crema: in 2018, more than 40 municipalities decided to invest in the realization of a traffic monitoring system in order to guarantee the safety of citizens in their territory. Consorzio.IT S.p.A., in collaboration with Maggioli Group, has created and developed an initiative called "Progetto Varchi" which comprises a vehicular traffic network [8].

The network is composed of 62 gates (63 in 2019), each controlled in both directions by two cameras, whose purpose is to register all the information about vehicles which pass through them and store it on a common database. Such a database is then connected to police systems, which are able to trigger alarms in real time, thereby identifying vehicles performing administrative as well as criminal violations.

My thesis provides an assessment of the potential of integrating advanced data analytics techniques in such a vehicular traffic network. Models and methods are designed, and experimented on real data kindly provided by the company.

In details, the goals of this thesis are:

- *Preliminary evaluation* of data and development of visualization tools for vehicle trajectories;
- *Descriptive analysis* of the system: development of a stochastic model and corresponding simulation;
- *Predictive analysis* of traffic on different kind of network links; profiling of key network points;
- *Prescriptive analysis* about the optimal distribution of police patrols with respect to the trajectories traversed by vehicles triggering alarms;
- *Experimental evaluation* of the aforementioned models.

In **chapter 2**, I describe the datasets' structure and my preliminary evaluation: it proved good quality in data, both in terms of completeness and accuracy. In particular,

I have built an effective visualization tool, relying on the use of GIS maps and reading of traffic data entries to display vehicles' paths on the map.

In **chapter 3**, I describe my descriptive analysis: I have built a model of the system which aggregates different random variables representing the arrival of vehicles in the system, their exit, their moving from one gate to another and the travelling speed. I have carried out a parameter estimation phase on real data, computing values such as spawn rate on each camera over time, mean time and the probability of transitions to mirror the whole traffic situation. Models and parameter values are included in an agent based framework, and finally in a simulation tool. As key performance indicators, I have chosen all the camera's transits in different timeslots and total number of transits in the whole map.

For most of the trials, the mean square error is low and the rate between real and simulated transits is near one.

In **chapter 4**, I describe the *predictive analysis* in which I have performed two tasks. First, I have analysed cluster tendency and actual gate cluster structures. I have created a program that computes all the cameras' features: their spawn rates, numbers of transits, probabilities of exiting from the system after passing through the cameras, the variances of the entrances and exits. I have then applied K-medoids and hierarchical clustering algorithms and I have represented the results on the GIS map, to better understand if there is a correlation between cluster classes and geographical position, such as highway or secondary streets' gates.

For some of the evaluated timeslots, there is a fine correlation with their position.

Second, I have explored time series analysis of traffic on links, evaluating models with double seasonality. I have initially created a program that computes, for each hour of the whole dataset, all the transits that start with the first route's camera and have as its immediate successor the second one. I have considered the busiest route in evening, morning and on the whole day. After this operation, I have run time series analysis choosing seasonality of day and week, considering a fixed amount of days. With the models produced by this analysis, I have predicted the traffic on certain weeks outside the training set, and I have compared predictions with the actual values from the real data.

For most of the chosen routes, the prediction was very accurate (in some cases, almost identical) with a small mean square error.

In **chapter 5**, I describe the *prescriptive analysis*: I have extracted from the dataset all the transits of vehicles that trigger at least one alarm. As examples, alarms are triggered if the vehicle is stolen, with expired insurance, with expired inspection or marked as suspicious by the police department. I have listed all the vehicles' daily paths and how many times each one of them is repeated. Using this information, I have compared two different approaches: an explicit integer programming model optimization, and the frequency based heuristic. The integer programming model determines the optimal position of a fixed number of patrols, in order to intercept the maximum number of vehicles producing alarms. The frequency based heuristic, instead, simply fixes patrols' positions near the gates with highest number of transits. I have compared the two solutions in terms of overall expected number of vehicles producing alarms which are intercepted. For small number of patrols, the two solutions are very similar, but for higher number of patrols the optimal solution keeps getting better.

In **chapter 6**, I present my conclusions and future implementations.

For what concerns technologies, I have used Java to develop several programs for extracting all the dataset features for the descriptive and prescriptive analysis. The Anylogic software has been used for descriptive modelling and agent based simulations. Data processing, clustering and time series analysis and prediction have been carried out by means of R scripts. Finally, for the prescriptive analysis I have implemented MathProg models, and used AMPL and IBM CPlex to obtain proven optimal solutions.

Figure 1.1: "Progetto Varchi" main banner

(a) Java (b) R (c) AMPL (d) Anylogic

Figure 1.2: Used technologies logos

Chapter 2

Data Structure

In this chapter, I describe the datasets kindly provided by Consorzio IT S.p.A. in terms of what are the parameters involved and what information is useful to the following analysis. Furthermore, I propose the development choices and preprocessing procedures applied to data.

I also present an effective visualisation tool that I have built: relying on the use of GIS maps, it reads traffic data entries to display vehicles' paths through real time on the map.

2.1 Datasets

The company has granted access to two different datasets which are a subset of their original ones: the first one contains all the information about gate's cameras, their position, active status and other parameters, the second one, instead, contains all the transits registered by each gate's cameras including time of arrival, vehicle's plate and type. For privacy reasons, the plate field has been hashed but the used function does not negatively influence the analysis, since for every plate, its hashed result is identical for each occurrence.

2.1.1 Gate Dataset

This set contains a line for each gate's camera installed in the city of Crema and neighbouring municipalities. The number of gates installed in 2018 was 62 and, in 2019, 63. Because each gate has mounted two different cameras, in order to check traffic from both directions, the total number of rows is 126. The attributes are described in Table 2.1.

There was no need to perform any preprocessing procedures on this dataset because data was complete and correct. For the following analysis I consider all gates and their most relevant features are: `id`, `gate_id`, `latitude` and `longitude`.

2.1.2 Traffic Dataset

This set contains a line for each transit of a vehicle through a camera. The dataset attributes are described in Table 2.2.

Name	Data Type	Description
id	integer	Primary key of this dataset, it is used to identify gates' cameras univocally
active	boolean	It denotes if the camera is currently monitoring traffic or if it's under maintenance
description	string	Textual description of the camera's position by providing a street name and/or highway code
gate_id	integer	Gate identifier
latitude	coordinate	Latitude of the gate
longitude	coordinate	Longitude of the gate
canCheckBothLanes	boolean	It denotes if the camera is able to monitor traffic for both sides of the road. Actually, none of the cameras installed is able to do it.
registrationDate	timestamp	Date and time when the camera was added to the system

Table 2.1: Camera dataset attributes

The Company provided us with all the transits of September 2018, May 2019, June 2019 and July 2019.

In September's dataset, the currently active cameras were 124 and, for the other datasets, they were 126. Furthermore, September's dataset is lacking all the alarm fields because, in that month, the company had not obtained permission, from the police department, to record that information yet.

2.1.2.1 Preprocessing

Before using this dataset, I have performed some data cleaning procedures. In some cases, the `vehicle_plate` field is empty and other important fields are missing or with unexpected values. Therefore, this transits are deleted from the dataset.

Since `date` field is composed by day and hour of the transit, I have decided to split it into two parts, `day` and `hour`, for better time manipulation in the following analysis.

After performing these procedures, I have sorted the dataset by three different columns: `vehicle_plate`, then `day` and finally `hour`. I performed this sorting, not only for performance purposes but also because it was necessary to see all the vehicle transits in order of occurrence for deducting drivers' possible paths and behaviours.

For the whole project, I have considered only their most relevant features which are `day`, `hour`, `vehicle_type`, `vehicle_plate`, `camera_id` and `allarm_type_id`.

The primary key of the dataset (`transit_id`) is not useful because `date` field has millisecond precision, so the group composed by `vehicle_plate`, `date` and `camera_id` is unique and it can be used as primary key.

Name	Data Type	Description
transit_id	integer	Primary key of the dataset, it is used to identify the transit univocally
date	timestamp	Timestamp representing the exact moment when the transit happened, in date and time format
day_of_week	integer	Day of the week in which the transit happens. It's an integer value between 1 (Sunday) and 7 (Saturday)
hour_of_day	integer	Hour of the day in which the transit occurs
vehicle_type	integer	Number representing vehicle's type that passed through the gate (car, truck, motorcycle, ...)
nation	string	It's the State in which the vehicle's plate is registered
vehicle_plate	string	Vehicle's plate of the transit: for privacy reasons, every plate has been hashed
trailer_plate	string	Vehicle trailer plate; if there is no trailer, the field is empty
kemler_code	string	Possible code attached to truck's plate in which they are carrying dangerous materials
camera_id	integer	Camera identifier that registers the transit
direction	string	It denotes the direction of the vehicle with respect to the camera orientation. It can be either APPROACH, GOAWAY or UNKNOWN
latitude	coordinate	Latitude of the transit camera
longitude	coordinate	Longitude of the transit camera
allarm_type_id	integer	Integer value representing possible administrative or criminal violations of the vehicle, such as stolen, with expired insurance or inspection. This field is repeated six times in the dataset because a vehicle may be able to trigger multiple alarms at the same time

Table 2.2: Traffic dataset attributes

2.2 Visualization

In order to fully understand the traffic situation of the Crema region, I have built an effective visualization tool in which, using vehicles' transits, I have plotted their possible path on a GIS (Geographic Information System) map.

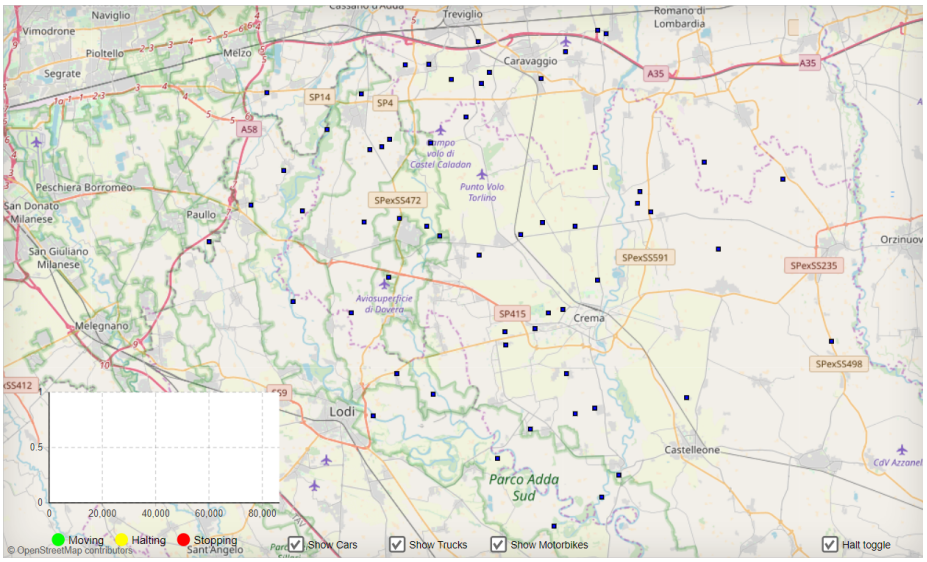
I have built this tool using Anylogic simulation software and, using this framework, I create an agent-based model because the core driver of the model are interactions between agents and in this case vehicles interact with gates' cameras.

2.2.1 Initialization and interface

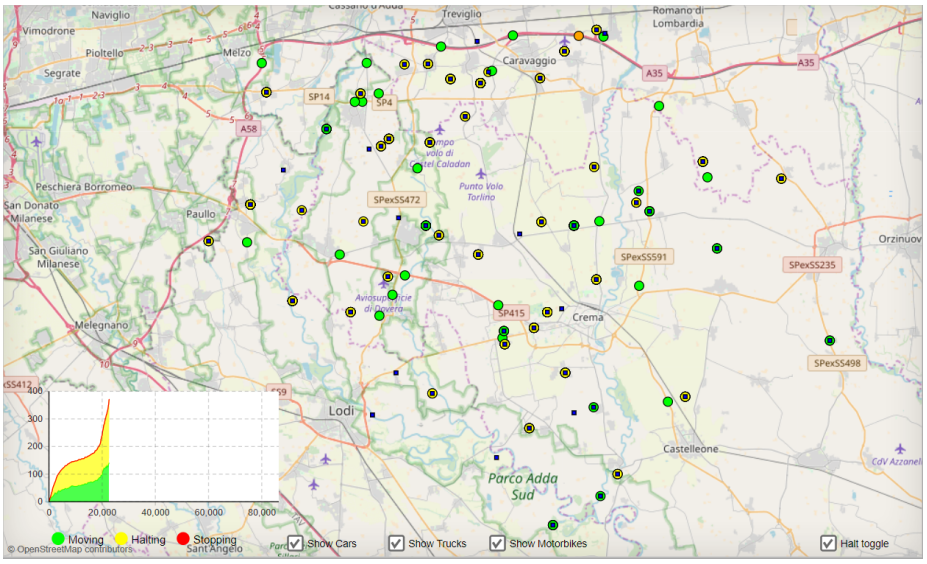
First of all, I have loaded gate dataset into the model, by creating a population of agents, and placed all of the gates on the map, considering their coordinates.

The interface shown in Figure 2.1, displays both the initial and running configuration of the tool in which gates, represented as blue squares, have been placed in their position.

For privacy and security reasons, the actual position of the cameras cannot be disclosed, therefore, in the figures, I have proposed a random distribution of the gates position.



(a) Initial configuration



(b) Running configuration

Figure 2.1: Visualization tool

I have also provided the possibility of showing only certain types of vehicle, by selecting the relative radio button. I have decided to implement an optional feature, that considers halt state of the vehicles which is explained in section 2.2.2.1. This feature can be toggled on or off at any time, by selecting the appropriate button. Furthermore, I have inserted

a time stack chart that displays, in real time, the number of vehicles moving (green), in halt state (yellow) and exited from the system (red).

2.2.2 Vehicle

At the beginning, I have created a population of agents to represent the vehicles. Each vehicle is composed by the following parameters: plate, vehicle type and a list containing all of its transits sorted by date and hour. Since my Anylogic version has limitations on the creation of agents, and due to overhead of the framework, I have decided to include in the system all the vehicles whose lists have 5 or more entries. Because the dataset contains only gates transits, it is impossible to know the exact journey made by each vehicle, therefore, I have used the route provider, included in Anylogic, which estimates the fastest route on real roads between two points on the map.

2.2.2.1 Halt feature

I have decided to implement this feature, because, in some cases, a vehicle could have two consecutive transits that spaced out between them for several hours and this means that the transit's speed is very low and makes the estimate less accurate.

To solve this problem, I decided to set a fixed speed threshold in which, all vehicles moving below this parameter are considered in halt state, and, all vehicles travelling faster, are considered in moving state. To get the travelling speed of a vehicle, I have computed the distance between two gates transits with Anylogic route provider, and I have also determined trip time by calculating the difference between first and second transit timestamps.

Once a vehicle is in halt state, it stays that way until the overall speed became greater than the given threshold and then vehicle can go into moving state and proceed to his next destination. The waiting time w of a vehicle in halt state, considering its two transit t_1 and t_2, is determined as follows:

$$w = (t_2.time - t_1.time) - \frac{dist(t_2.gate, t_1.gate)}{speedThreshold}$$

As mentioned before, this feature can be disabled: in this case, speed threshold is set to zero.

2.2.2.2 Behaviour

Vehicles are represented in the map as circle and have different colours based on their status:

initial the vehicle has been created and it has been positioned on the map. In this phase, its colour is white;

moving the vehicle is moving from a gate to another, the colour in this phase is different for each kind of vehicles: green is used for cars, orange for trucks and purple for motorbikes;

halt if the halt feature has been enabled, the vehicle is in halt state and it stops at a gate's position until it's ready to move again. In this phase, its colour is yellow.

stop the vehicle has completed its journey by travelling for all of the transits contained in its transit list and then it's deleted from the system. In this phase, its colour is red but it's not visible since the vehicle has been removed from the map.

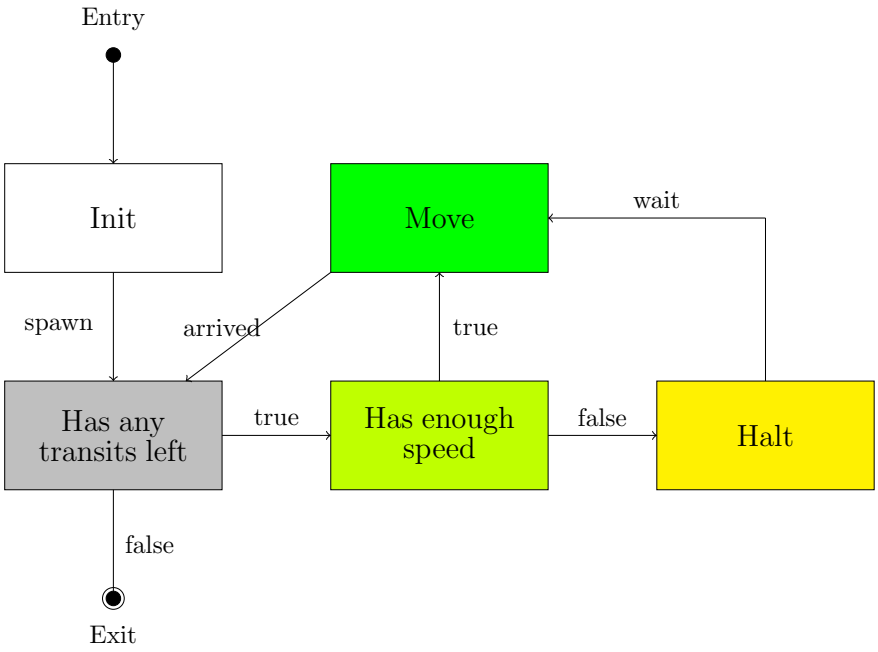


Figure 2.2: Vehicle behaviour schema

Using the framework, I have modelled the behaviour of each vehicle as shown in Figure 2.2 and it's composed by the following steps:

1. After creating the vehicle, It is spawned and added to the map at its first transit's camera location;

2. it checks if the vehicle has any transits left in his list, if it does, go to step 3; otherwise the vehicle's journey has ended and it's removed from the system;

3. if the halt feature has been enabled, it checks vehicle speed and decide if the vehicle is considered moving or in halt state. If speed is enough or the halt feature is disabled then go to step 5, otherwise go to step 4;

4. the vehicle stops at his current camera location, change its colour and wait until his speed become sufficient to be considered moving, then go to step 5;

5. the vehicle moves from his current position to the next transit's gate location. Transit duration is calculated as the difference between current and next transit's timestamps. After the vehicle has arrived at its destination, go to step 2.

2.2.3 Gate camera

Each gate camera is composed by the following parameters: camera id, gate id, latitude and longitude.

For every gate camera, I have monitored vehicles transits by measuring three different situations: how many vehicles started their journey in this camera, how many of them are in halt state, and, how many of them have terminated their trip at this point. These features are represented with a bar plot as shown in Figure 2.3 These counters are constantly updated during the whole presentation.

This information is very useful to distinguish access gates, in which most of his passing vehicles start their journey there, from exit gates, in which most of his passing vehicles end their journey there.

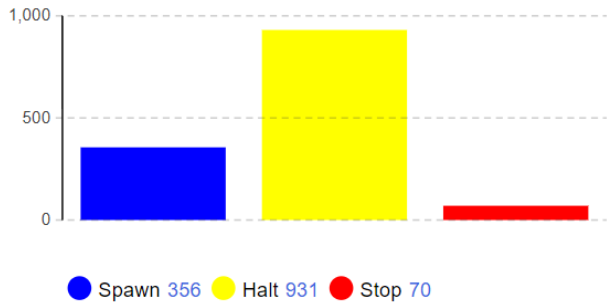


Figure 2.3: Bar plot of a gate camera

Chapter 3

Descriptive Model

In this chapter, I describe my descriptive analysis: I have built a model that is able to simulate the traffic situation by computing and aggregate different parameters calculated using the real transits. I also compare this simulated situation with the real traffic distribution by measuring some KPIs.

3.1 Parameters estimation

Before building the model, I have computed different useful parameters, estimated from the real transits data, in order to build a descriptive model able to best represent the traffic situation. In this model I have considered the following components:

spawn rate probability of a vehicle to begin its journey at the position of a specific gate camera in the system;

travelling path probability of a vehicle currently located at gate camera i to travel towards the position of gate camera j;

travelling time mean trip time of a vehicle located at gate camera i to travel towards the position of gate camera j;

exit probability probability of a vehicle currently located at the position of a specific gate camera to end there its journey.

3.1.1 Time division

Since traffic distribution is not regular over time, it is unlikely that a single parameter is capable of representing the traffic situation of a full day, but it's reasonable to think that in certain part of the day, or certain days over a week, traffic may have regularities.

After visualising the traffic situation, described in section 2.2, I have observed that, for example, morning traffic is different from evening or afternoon traffic and also Wednesday vehicles distribution is different from Sunday's. Therefore I have decided to perform two time splits, one for weekdays and the other for each part of the day.

3.1.1.1 Week division

After performing some preliminary analysis, I have decided to split the week into two categories, called *day types*: one for workdays which ranges from Monday to Friday included, and the other for holidays, which is composed by Saturday and Sunday.

I have decided to apply this split because I have noticed that for both of the two categories the difference between each one of their days is negligible.

3.1.1.2 Day division

After performing some preliminary analysis, I have decided to split the day into 9 categories, called *time slots*, which are listed in Table 3.1.

Time Slot	**From**	**To**
F0005	00:00:00	04:59:59
F0506	05:00:00	05:59:59
F0607	06:00:00	06:59:59
F0709	07:00:00	08:59:59
F0913	09:00:00	12:59:59
F1317	13:00:00	16:59:59
F1719	17:00:00	18:59:59
F1922	19:00:00	21:59:59
F2200	22:00:00	23:59:59

Table 3.1: Day division: *time slots*

Considering both splits, the total number of division applied is $2 \times 9 = 18$.

3.1.2 Spawn rate

Spawn rate of a gate camera is the probability that a vehicle begin its journey at the camera location. Since spawns of vehicles in the system can be computed as inter-arrival times, the probability distribution that I have chosen to model this phenomenon is the Negative Exponential Random Variable and the only parameter to estimate is the rate λ.

I have used a dataset containing lots of entries so, according to the theory, λ can be determined by counting all the occurrences of vehicles that spawn in a given gate camera, on a given time slot and day type, and dividing it by the elapsed time. My lambda is expressed as number of vehicles per hour. The elapsed time is determined by multiplying the duration, in hour, of the given time slot and the number of days of the given day type in which this phenomenon has been monitored.

The result is a 3-dimensional array $\Lambda_{i,j,k}$, where i denotes the day type, j the time slot and k the camera id.

I have implemented this procedure as a Java application using the sorted traffic dataset, and applying the following steps:

1. I have taken the first entry of each vehicle and, using this transit, I have extracted the relative day type, time slot and camera id;

2. With this information, I have incremented by one a 3-dimensional array, whose purpose is to count all the transits of that particular gate camera in that particular time slot;

3. After scanning the whole dataset, I have computed lambda $\Lambda_{i,j,k}$ for each day type, timeslot and gate camera by dividing the counter with its elapsed time (*hourpassed*).

The pseudo-code of this procedure is shown in Algorithm 1.

Algorithm 1 Spawn rate algorithm

▷ Counting all occurrences
for all $line \in Dataset$ **do**
  if isFirstTransit($line.vehicle\_plate$) **then**
    $Counter_{type,slot,camera} \leftarrow Counter_{type,slot,camera} + 1$
  end if
end for
▷ Computing spawn Rate
for all $type \in DayTypes$ **do**
  for all $slot \in TimeSlots$ **do**
    for all $camera \in \{i | 1 \leq i <= NumberOfCameras\}$ **do**
      $hourspassed \leftarrow$ duration($slot$) $\times$ DayPassed($type$)
      $\Lambda_{type,slot,camera} \leftarrow \frac{Counter_{type,slot,camera}}{hourspassed}$
    end for
  end for
end for

3.1.3 Travelling parameters

In order to model the journey of a vehicle, I need to compute a parameter that is able to estimate, based on the vehicle's starting point, its next destination. Therefore, I have computed a 4-dimension origin-destination matrix $P_{d,ts,i,j}$, considering each day type d and each time slot ts, which represent the probability of a vehicle currently at gate camera i to travel towards gate camera j;

Since each route has different distance and speed limit, it is unlikely that all of the possible routes have the same travelling time. In order to create an accurate model, I have also computed, for each day type and time slot, the mean time needed to travel from gate camera i to gate camera j.

The results are two 4-dimensional arrays: one is $P_{d,ts,i,j}$, which represents the origin-destination matrix, and the other is $T_{d,ts,i,j}$, which contains the route mean travelling times.

I have implemented this procedure as a Java application using the sorted traffic dataset, and applying the following steps:

1. I have analyses the transits in pairs and, if they have the same plate, day and time slot, I have incremented by one the corresponding counter considering the first transit camera of the pair as origin and the second transit gate camera as destination. Meanwhile I have computed the travelling time between the two transits and add the result to another counter;

2. After scanning the whole dataset, I have computed, for each day type, timeslot and routes the two results. For what concerns the origin-destination matrix $P_{d,ts,i,j}$, I have divided the relative counter with the total number of the transits of the first camera i (Normalization process). For what concerns the travelling times matrix $T_{d,ts,i,j}$ I have divided the relative time counter with the passages counter in order to compute the mean.

The pseudo-code of this procedure is shown in Algorithm 2.

3.1.4 Exit probability

Spawn and travelling parameters are not sufficient to describe the whole traffic system as the vehicles would remain in the system indefinitely. Therefore it is necessary to compute a parameter that predict, once a vehicle is at a gate camera location, if the journey can continue or be interrupted. Since the outcome is binary, I have chosen the Bernoulli Random Variable [9].

The result is a 3-dimensional array $B_{i,j,k}$, that contains the probability of ending the journey, where i denotes the day type, j the time slot and k the camera id.

I have implemented this procedure as a Java application using the sorted traffic dataset, and applying the following steps:

1. For each vehicle's journey, I have considered their last transit and retrieve its day type, time slot and camera id;

2. With this information, I have incremented by one a 3-dimensional array, whose purpose is to count all the transits of that particular gate camera in that time slot. Meanwhile I have also counted all the transits happened for each camera, in every day type and time slots;

3. After scanning the whole dataset, I have computed the Bernoulli probability $B_{i,j,k}$ for each day type, timeslot and gate camera by dividing the counter with the total number of passages.

The pseudo-code of this procedure is shown in Algorithm 3.

3.2 The model

The purpose of this model is to simulate, without the real data, the traffic situation using the parameters calculated in the previous section. To build and simulate this model, I have used the Anylogic framework, in particular the GIS maps and route provider as I have done in section 2.2.

Algorithm 2 Travelling parameters algorithm

▷ Counting all occurrences
$previous \leftarrow NULL$
for all $line \in Dataset$ **do**
  if $previous \neq NULL$ **then**
    if $previous.vehicle\_plate = line.vehicle\_plate$ **then**
      if $previous.date = line.date$ **then**
        if IsSameTimeSlot(previous, line) **then**
          $Counter_{type,slot,from,to} \leftarrow Counter_{type,slot,from,to} + 1$

          $travelTime \leftarrow \text{Duration}(previous, line)$
          $Time_{type,slot,from,to} \leftarrow Time_{type,slot,from,to} + travelTime$
        end if
      end if
    end if
  end if
  $previous \leftarrow line$
end for
▷ Computing travelling parameters
for all $type \in DayTypes$ **do**
  for all $slot \in TimeSlots$ **do**
    for all $camera_i \in \{i | 1 \leq i <= NumberOfCameras\}$ **do**
      $sum \leftarrow \text{SumRow}(camera_i)$
      for all $camera_j \in \{j | 1 \leq j <= NumberOfCameras\}$ **do**
        $T_{type,slot,camera_i,camera_j} \leftarrow \frac{Time_{type,slot,camera_i,camera_j}}{Counter_{type,slot,camera_i,camera_j}}$
        $P_{type,slot,camera_i,camera_j} \leftarrow \frac{Counter_{type,slot,camera_i,camera_j}}{sum}$
      end for
    end for
  end for
end for

Algorithm 3 Exit rate algorithm

▷ Counting all occurrences
for all $line \in Dataset$ **do**
  if isLastTransit($line.plate$) **then**
    $Counter_{type,slot,camera} \leftarrow Counter_{type,slot,camera} + 1$
  end if
  $totalTransits_{type,slot,camera} \leftarrow totalTransits_{type,slot,camera} + 1$
end for
▷ Computing exit Rate
for all $type \in DayTypes$ **do**
  for all $slot \in TimeSlots$ **do**
    for all $camera \in \{i | 1 \leq i <= NumberOfCameras\}$ **do**
      $B_{type,slot,camera} \leftarrow \frac{Counter_{type,slot,camera}}{totalTransits_{type,slot,camera}}$
    end for
  end for
end for

Since the parameters estimation phase, that I have performed, split time into 18 parts, I have included in the framework a global variable containing all of the computed values for every part. While the simulation proceeds, in order to properly use the correct parameters, I have set a global event, repeated every hour, responsible to properly switch both day type and time slot according to the simulation date and time, so that agents can access the information they need without worrying about continually checking the correctness.

For what concerns the origin destination matrix, I have implemented an inverse transform distribution function: after computing the cdf (cumulative distribution function) for each gate camera, which correspond to a row of the matrix, I have created a `get` method that randomly chose a camera based on the computed cdf[7].

For what concerns spawn rate, mean travelling time and exit probability, I have included them without further transformation.

At the end of the simulation, the tool produces a CSV file that contains the number of all passages monitored by each gate camera. This file is used to apply the accuracy evaluation of the model described in section 3.3.

3.2.1 Gate camera

I have loaded gate dataset into the model, by creating a population of agents, and placed all of the gates on the map, considering their coordinates. Each agent is composed by the following parameters: camera id, gate id, latitude, longitude and a counter, whose purpose is to count all the passages monitored by this camera.

Since vehicles' journey always begins at a gate camera location, I have assigned, to each gate cameras, an event that dynamically creates a new vehicle based on the current λ of the spawn rate parameter.

3.2.2 Vehicle

I have created a population of agents to represent the vehicles. Each vehicle is composed by only one parameter:the spawn camera, which is set by the gate camera that creates the vehicle. They also refer to the same global variable that contains all the origin-destination matrix, the travelling time matrix and the exit probabilities. Based on their current location, they access the appropriate structure they need in order to perform their interactions.

In this case, I haven't considered halt state of the vehicles and their graphical representation is a green circle.

Using Anylogic, I have modelled the behaviour of each vehicle as shown in Figure 3.1 and it's composed by the following steps:

1. after the gate camera creates the vehicle, It is spawned and added to the map at the gate camera location;

2. it check the exit probability parameter, if the vehicle has ended its journey or it can move to another location, if the outcome is positive, then the vehicle is removed from the system; otherwise go to step 3;

3. the vehicle call the `get` method of the inverse transform distribution function and, according to its current location, it chose his next gate camera destination;

4. the vehicle moves from his current position to the next calculated gate location. Transit duration is retrieved from the mean travelling time matrix. After the vehicle has arrived at its destination, go to step 2.

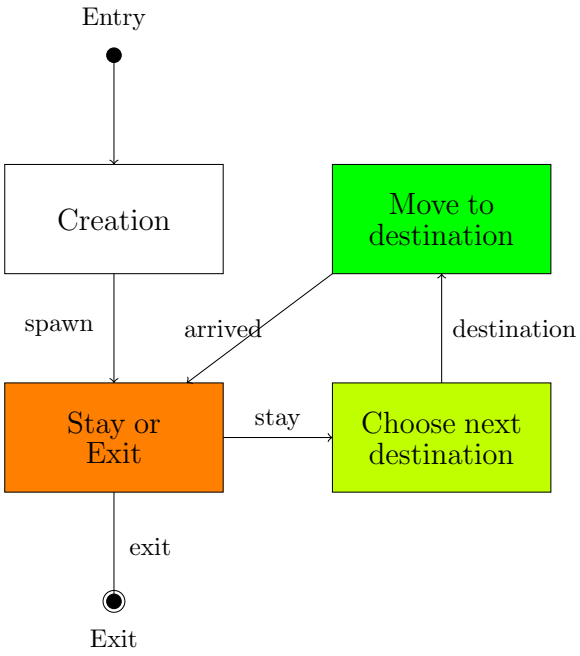


Figure 3.1: Vehicle behaviour schema

3.3 Experiments

In order to evaluate the accuracy of the created descriptive model, I have performed different experiments. The experiment consists in comparing the data generated by the simulation with the real transit dataset. In this regard I have chosen two key performance indicators: the first is a *list* containing all of the transits for each room, and the second is the *ratio* between the number of total simulated transits and the real one. Furthermore I have also determined the mean square error between the two distributions.

In order to retrieve the KPIs from the real transit dataset, I have created a Java application that, given the time range, extract all the information needed.

To better understand the shape of the two distributions, I have computed both Probability Mass Function (pmf) and Cumulative Density Function (cdf). Given the distribution $dist$ that contains all the transits for each of the N gate cameras, the probability mass function pmf is determined this way:

$$pmf(i) = \frac{dist(i)}{\sum_{j=1}^{N} dist(j)}, \quad \forall i \in 1 \leq i \leq n$$

Given the probability mass function pmf calculated on the distribution $dist$, the cumulative density function cdf is determined this way:

$$cdf(i) = \begin{cases} dist(i) & \text{if } i = 1 \\ dist(i) + cdf(i-1) & \text{if } 2 \leq i \leq N \end{cases}$$

I have performed 4 experiments by simulating different parts of a random workday.

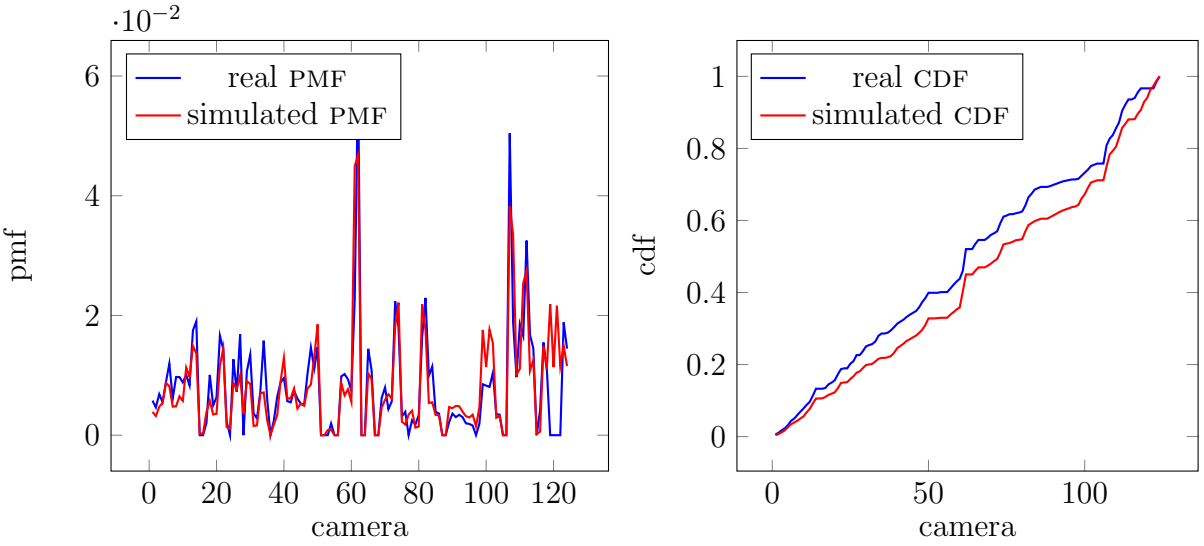
After computing the two functions, I have plotted them on the same graph and notice their similarity. I have also computed both ratio and the two mean square error (MSE). For what concerns *afternoon*, *evening* and *full day* experiments, the two distributions have almost the same number of transits, hence ratio is near 1. In both cases, the mean square errors are very low. For what concerns *morning* experiment, the ratio is decent but the mean square errors are slightly higher.

After completing these experiments, considering the results of the KPIs and the errors, I have proved that the created model is very accurate.

The results are presented in Table 3.2 and the graphical representations of the two functions are displayed in Figures 3.2, 3.3, 3.4 and 3.5.

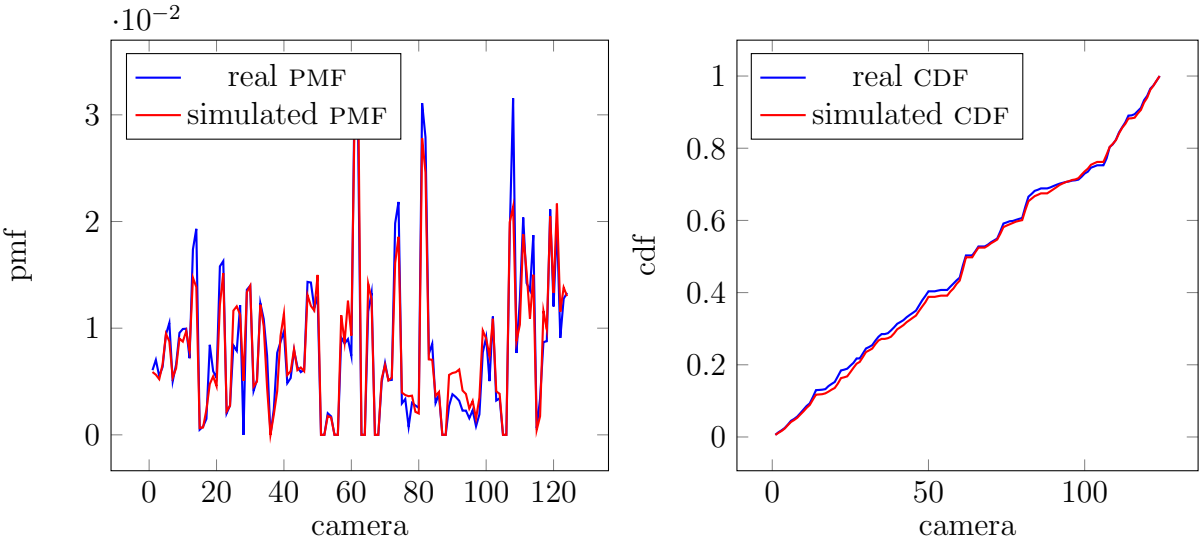
Experiment	**Range**		**Transits**		**Ratio**	**MSE**	
Name	**start**	**end**	**Real**	**Simulated**		**pmf**	**cdf**
morning	00	08	68399	42585	0.62255	0.000027	0.003725
afternoon	13	19	169402	118174	0.69759	0.000004	0.000110
evening	19	23	55757	44411	0.79650	0.000009	0.000180
full day	00	00	407737	440152	1.07949	0.000005	0.000272

Table 3.2: Experiment for descriptive model



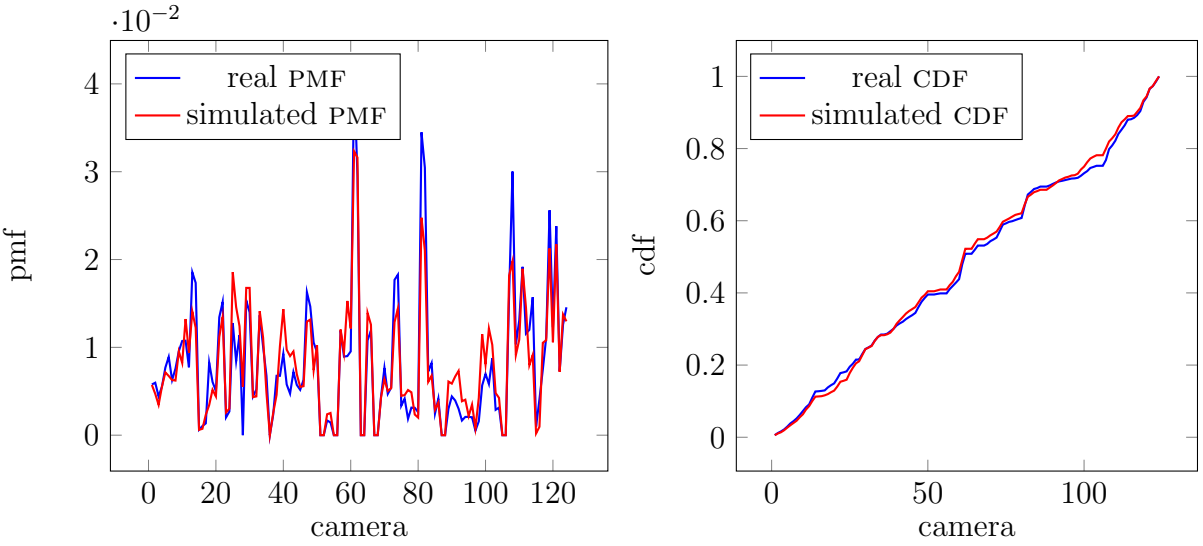
(a) Probability Mass Function (b) Cumulative Density Function

Figure 3.2: Graphical representation of PMF and CDF of *morning* experiment



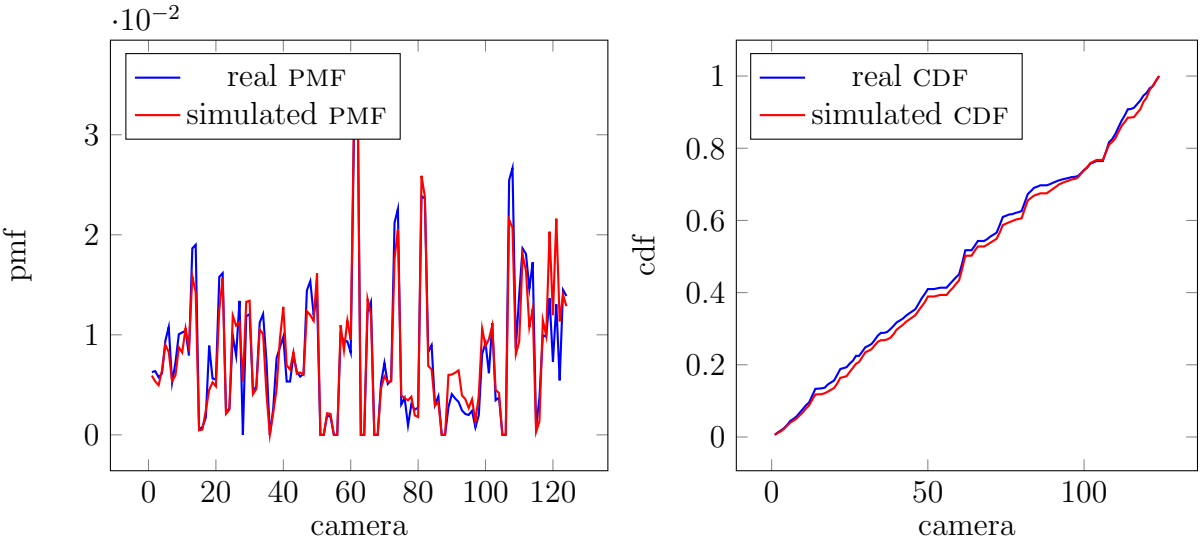
(a) Probability Mass Function (b) Cumulative Density Function

Figure 3.3: Graphical representation of PMF and CDF of *afternoon* experiment



(a) Probability Mass Function (b) Cumulative Density Function

Figure 3.4: Graphical representation of PMF and CDF of *evening* experiment



(a) Probability Mass Function (b) Cumulative Density Function

Figure 3.5: Graphical representation of PMF and CDF of *full day* experiment

Chapter 4

Predictive Models

In this chapter, I describe the two task of my predictive analysis. In the first task, I have analysed cluster tendency and actual gate cluster structures by applying different clustering algorithms and comparing their features. In the second task, I have explored time series analysis of traffic on links, evaluating models with double seasonality and then comparing the resulted model with the actual traffic.

4.1 Gate Cluster Analysis

4.1.1 Theoretical introduction

Clustering is the process of grouping a set of elements in containers, called *clusters*, which are *similar* to one another, by having high intra-class similarity, and *dissimilar* to the objects in other groups, by having low inter-class similarity. This procedure is useful, especially in data mining, to identify distribution patterns and correlation between objects.

There are several clustering technique. In this section I describe the ones used for this analysis:

K-mean given k as input, partition a set of objects into k clusters by minimizing the mean square error between object in the same cluster, and using as reference point the mean value of these objects;

K-medoids given k as input, partition a set of objects into k clusters by minimizing the mean square error between object in the same cluster, and using as reference point a *medoid*, which is the most centrally located object in the cluster. The algorithm that I have considered is Partitioning Around Medoids (PAM);

Hierarchical Clustering group data objects into a tree of clusters. It can proceed in two different ways: *agglomerative*, also called bottom-up, is the approach in which starts with each object forming a separate group and then merge the group close to one another until a stop condition is met; instead *divisive*, also called top-down, start with all the objects in the same cluster and then split it into smaller clusters until a stop condition is met [4].

4.1.2 Features extraction

Before using the clustering algorithms, I have created a Java application that extracts the desired features of each gate camera from the traffic dataset, specifically the data of September 2018. Considering that it is unlikely that every gate camera observed the same behaviour through the whole day, a single set of features would have not been very accurate. Therefore, I have applied the same time division used in the descriptive model, described in section 3.1.1.

For each gate camera i, in each time slot, I have computed the spawn rate λ (section 3.1.2), the exit probability B(section 3.1.4) and the N total number of transits. Considering the origin-destination matrix, described in section 3.1.3, I have computed the entrance variance σ^2_{in}, which is calculated considering the set of probabilities of arriving at gate camera i, and the exit variance σ^2_{out}, which is calculated considering the set of probabilities of exiting from gate camera i.

Since, in this analysis, I have considered gates instead of cameras, I would have had two parameters for each of them. Instead of excluding one over the other I have decided to sum the two camera parameters and using them as a single gate feature.

In table 4.1, I present the computed features considering a timeslot which starts at hour t_s and end at hour t_e.

Symbol	**Name**	**Computation**	**Description**
λ	spawn rate	$\lambda = \frac{duration(t_s,t_e)}{N}$	Probability, expressed as $\frac{vehicles}{hour}$, that a vehicle begin its journey at the gate location
B	exit probability	$B = \frac{\sum_{t_s}^{t_e} exit_i(t)}{N}$	Probability of ending the journey at the gate location
N	transits	$N = \sum_{t=t_s}^{t_e} transits(t)$	Number of transits of the gate
σ^2_{in}	varEnter	$\sigma^2_{in} = \frac{\sum_{t_s}^{t_e}(probEntering(t)-\mu)^2}{numberOfGates}$	Variance of the distribution composed by the probability of reaching the gate.
σ^2_{out}	varExit	$\sigma^2_{out} = \frac{\sum_{t_s}^{t_e}(probExiting(t)-\mu)^2}{numberOfGates}$	Variance of the distribution composed by the probability of leaving from the gate.

Table 4.1: Cluster features of each gate

4.1.3 The model

In order to create this model, I have used different R libraries. First, I have loaded the computed feature into the framework. Then, I have determined the optimal number of clusters by applying the silhouette method and computing the k-mean algorithm [5]. I have noticed that all the distributions have several outliers, probably because there were some switched-off gate cameras. Unfortunately, the resulted clusters were not very accurate, because the presence of outliers in a set of objects is one of the weakness of the k-mean algorithms.

Therefore, I have decided to apply the hierarchical algorithm, which is more resistant to outliers. The problem is that for the hierarchical clustering, an algorithm able to identify the optimal number of clusters has not been implemented yet. To overcome this problem, I have decided to use the optimal number of clusters obtained by applying the k-medoids algorithm.

The function for computing the k-medoids is provided by R libraries and it's called `pamk`. After computing the optimal k, I have applied the hierarchical clustering function, called `hclust`. The result is a tree, in which the leaves correspond to the gates. I have performed the cluster divisions with the function `cutree`, which, based on the k number of cluster, groups the tree branches together, and labels each group with an ordinal number that ranges from 1 to k included.

The pseudo-code of this model is shown in Algorithm 4.

Algorithm 4 Cluster procedure

```
1: for all file ∈ folder do
2:     featureSet ← READFILE(file)
3:                                              ▷ Computing optimal k
4:     pamResult ← PAMK(featureSet)
5:     optimalK ← pamk.nc
6:                                              ▷ Appling Hierarchical Algorithm
7:     treeResult ← HCLUST(featureSet)
8:     result ← CUTREE(treeResult, optimalK)
9: end for
```

4.1.4 Experiments

After applying the model to the computed gate features, the resulted clustering was fine for most of the timeslots. In some cases, however, the clustering division has produced unsatisfactory results, either for the low number of clustering or for the poor correlation between the gates in the cluster group.

I have decided to analyse two different timeslots of the two day types: a holiday in slots 9-13 and 19-22 and a workday in slots 7-9 and 17-19. I have created a visualization tool by using the GIS maps of Anylogic, in which I plot the gate on the map, represented as squares, and, based on their cluster number, I have provided a different icon colour for each of them.

By doing this representation, I was able to carry out an empirical geographic evaluation, trying to notice if the gates belonging to the same group were correlated with their position on the map, for example, the access points to the city of Crema or the gates near the high roads and so on. The graphical representation of the two day types are shown in Figures 4.1 and 4.2. As mentioned in section 2.2, for privacy and security reasons, I have randomized each gate actual position.

I have compared the features of each gate in the same group in order to understand the reason why they have been grouped together. After the comparisons, I have noticed that the most relevant feature, considered by the splitting procedure of the hierarchical

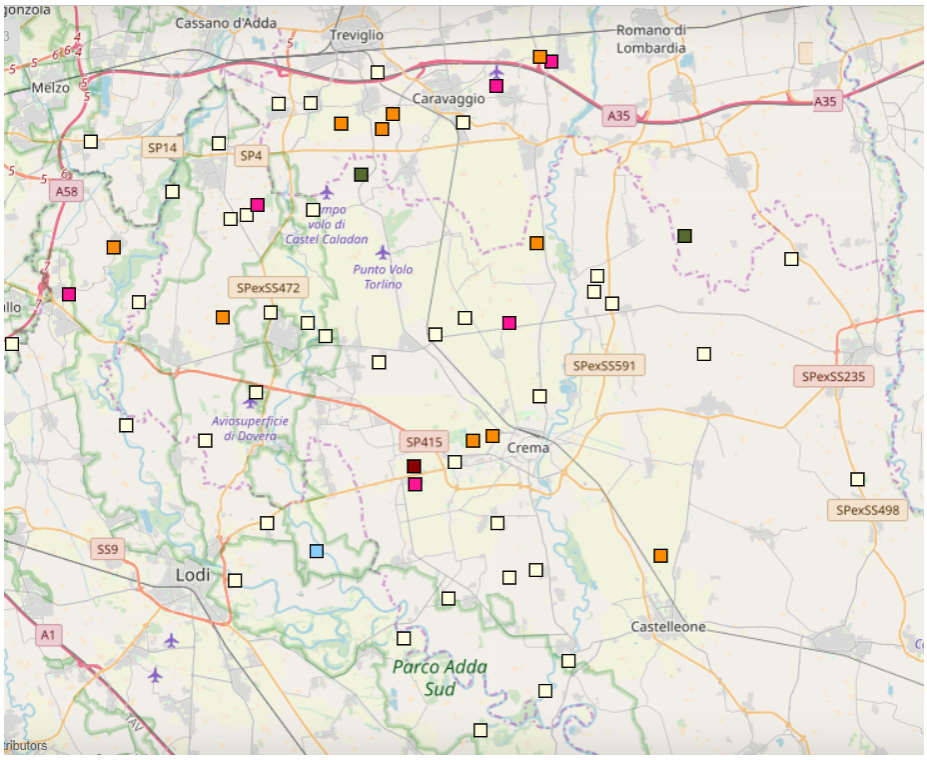
clustering algorithm, was the number of transits since almost every cluster group of each considered situations manifest this characteristic. The comparisons are shown in Tables 4.2, 4.3, 4.4 and 4.5.

Cluster		**Similar features**				
N	**colour**	$\boldsymbol{\lambda}$	**Exit** $\boldsymbol{p}$	**Transits**	$\boldsymbol{\sigma^2_{in}}$	$\boldsymbol{\sigma^2_{out}}$
1	white	< 0.5	-	-	$0.04 \leq \sigma^2_{in} \leq 0.09$	-
2	orange	-	$0.3 < p < 0.4$	≈ 15000	-	-
3	pink	-	< 0.3	≈ 26000	$0.04 \leq \sigma^2_{in} \leq 0.06$	-
4	red	-	-	< 1000	-	-
5	green	-	-	≈ 50000	≈ 0.007	-
6	cyan	-	-	≈ 38000	-	-

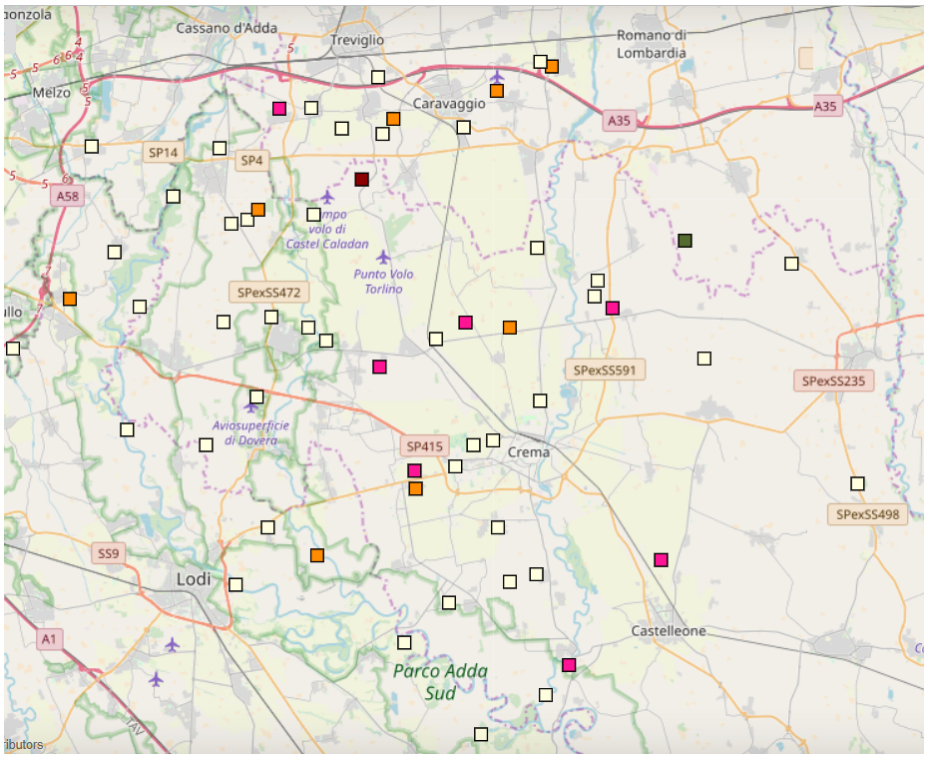
Table 4.2: Similarity in cluster for Holiday in time slot 9 - 13

Cluster		**Similar features**				
N	**colour**	$\boldsymbol{\lambda}$	**Exit** $\boldsymbol{p}$	**Transits**	$\boldsymbol{\sigma^2_{in}}$	$\boldsymbol{\sigma^2_{out}}$
1	white	-	< 0.5	-	$0.04 \leq \sigma^2_{in} \leq 0.09$	-
2	orange	-	< 0.4	≈ 15000	-	-
3	pink	-	< 0.3	≈ 26000	$0.06 \leq \sigma^2_{in} \leq 0.08$	$0.055 \leq \sigma^2_{out} \leq 0.075$
4	red	-	-	< 1000	-	-
5	green	-	-	≈ 50000	≈ 0.007	-
6	cyan	-	-	≈ 38000	-	-

Table 4.3: Similarity in cluster for Holiday in time slot 19 - 22

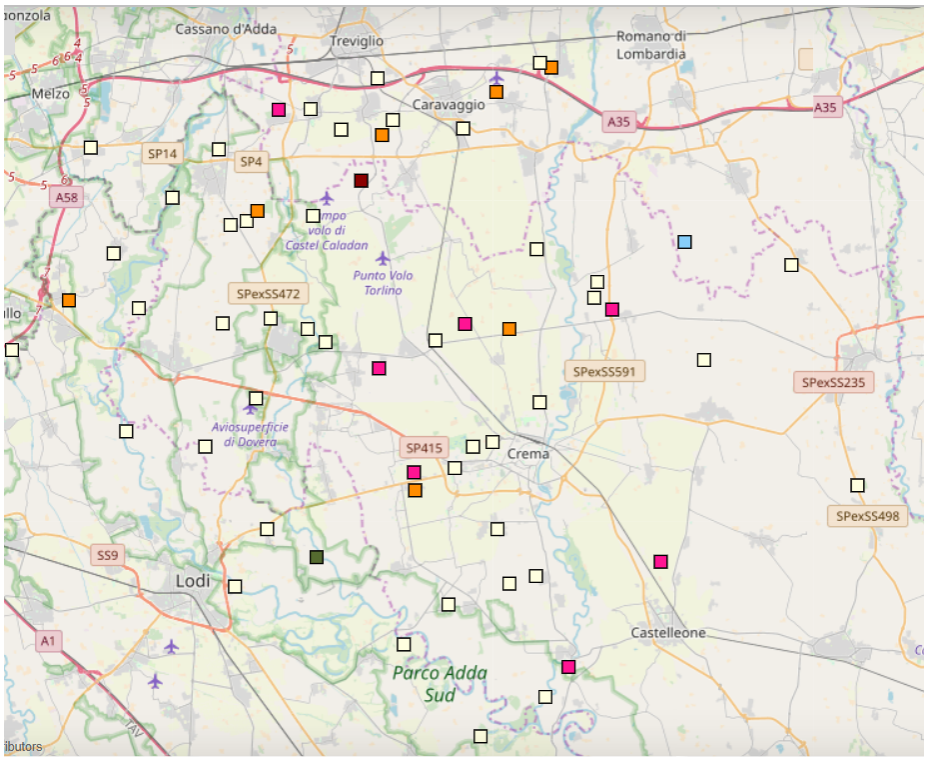


(a) time slot 9 - 13

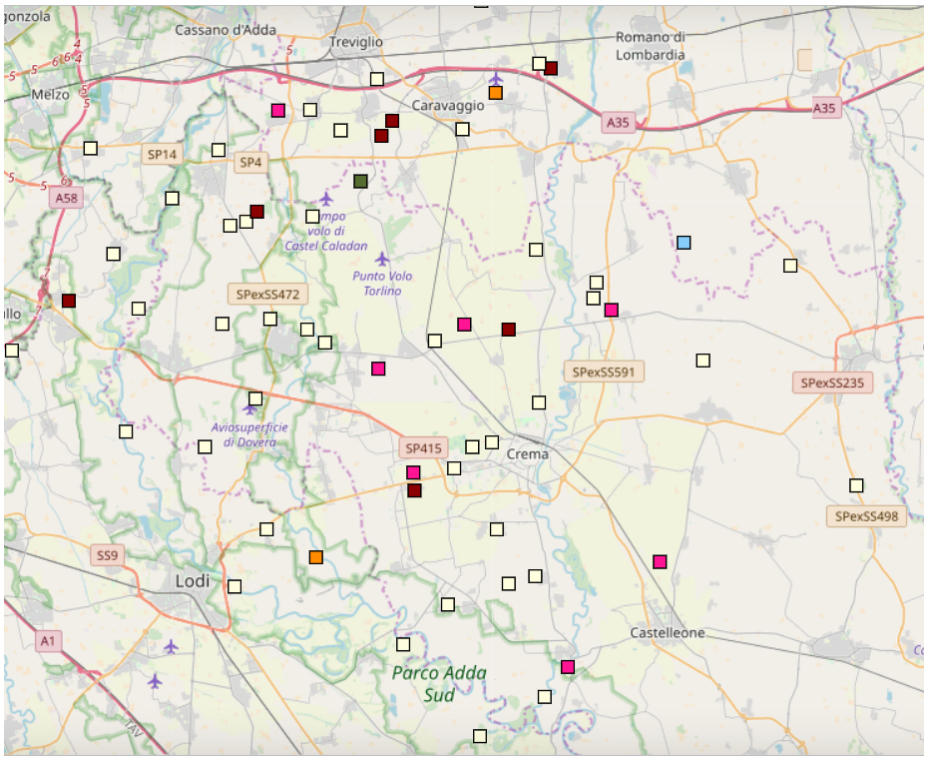


(b) time slot 19 - 22

Figure 4.1: Holiday geographical clustering representation



(a) time slot 7 - 9



(b) time slot 17 - 19

Figure 4.2: Workday geographical clustering representation

Cluster		**Similar features**				
N	**colour**	$\boldsymbol{\lambda}$	**Exit** $\boldsymbol{p}$	**Transits**	$\boldsymbol{\sigma^2_{in}}$	$\boldsymbol{\sigma^2_{out}}$
1	white	< 0.2	-	$10000 < T < 30000$	-	-
2	orange	-	-	≈ 42000	-	$0.050 \leq \sigma^2_{out} \leq 0.085$
3	pink	-	< 0.2	-	-	-
4	red	-	-	one entry	-	-
5	green	-	-	one entry	-	-
6	cyan	-	-	≈ 67000	-	-

Table 4.4: Similarity in cluster for Workday in time slot 7 - 9

Cluster		**Similar features**				
N	**colour**	$\boldsymbol{\lambda}$	**Exit** $\boldsymbol{p}$	**Transits**	$\boldsymbol{\sigma^2_{in}}$	$\boldsymbol{\sigma^2_{out}}$
1	white	< 0.3	-	$15000 < T < 30000$	-	-
2	orange	-	-	≈ 55000	-	$0.04 \leq \sigma^2_{out} \leq 0.06$
3	pink	-	< 0.2	-	-	-
4	red	-	< 0.25	≈ 42000	-	$0.050 \leq \sigma^2_{out} \leq 0.065$
5	green	-	-	one entry	-	-
6	cyan	-	-	≈ 80000	-	-

Table 4.5: Similarity in cluster for Workday in time slot 17 - 19

4.2 Route Time Series Analysis

4.2.1 Theoretical Introduction

A time series is a sequence of values y_t taken by a quantity of interest at given points in time t. Time series are used in pattern recognition, statistics, signal processing, mathematical finance, control engineering, weather forecasting, earthquake prediction, astronomy, communication engineering or in any field involving temporal measurement [2].

Time series analysis is a method for analysing the data to derive a meaningful statistics and characteristics of data in order to predict future trends. The observed values y_t of a time series are the result of a combination of several components of different nature such as:

- long period trend, m_t;
- long term economic cycles, v_t;
- seasonal component, s_t, given a specific period L;
- random residual, r_t.

4.2.2 Time series extrapolation

Before applying the predictive model, I have developed a Java application that, given a specific link between two gate cameras, determinates the complete time series, which corresponds to the number of transits happened in that route in a specific time. I have decided to count all the transits for each different hour of each day included in the given dataset, so, suppose that the dataset contains all the transits ranging for n number of days, the calculated series is composed of $24 \times n$ entries.

4.2.2.1 The procedure

After loading the dataset into the tool, I have performed a preliminary scan to list all the monitored days and sort them. As mentioned before, the time series is calculated considering a route between two gate camera R_1 and R_2 After performing this operation, I have analysed the transits in pairs and, if they have the same plate, I have checked if the first transit camera corresponds to R_1 and the second one corresponds to R_2, then if they transits happened in the same day and hour, I have incremented by one the counter related to the transits' day and hour.

The result is a 3-dimensional array, that contains all the transits happened in every hour of every single day.

The pseudo-code of this procedure is shown in Algorithm 5.

4.2.3 The model

The purpose of this model is to predict, without the real data, the route traffic of a set period of time using the time series calculated in the previous section. To build and run this model, I have implemented it in R Studio and I have used different R libraries.

Algorithm 5 Time Series extrapolation algorithm

```
1:                                                    ▷ Counting all passed days
2: for all line ∈ Dataset do
3:     if isAlreadyPresent(line.day) then
4:         DayCounter.add(line.day)
5:     end if
6: end for
7: DayCounter.sort()
8:                                                    ▷ Creating time series
9: previous ← NULL
10: for all line ∈ Dataset do
11:     if previous ≠ NULL then
12:         if previous.vehicle_plate = line.vehicle_plate then
13:             if EqualsChosenRoute(previous, line) then
14:                 if previous.date = line.date then
15:                     if previous.hour = line.hour then
16:                         position ← DayCounter.get(date, hour)
17:                         TimeSeries_position ← TimeSeries_position + 1
18:                     end if
19:                 end if
20:             end if
21:         end if
22:     end if
23: end for
```

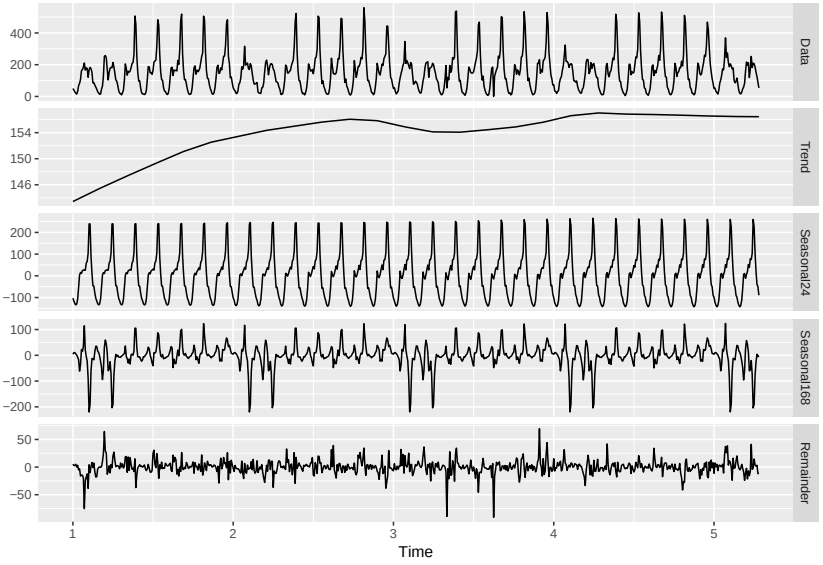


Figure 4.3: An example of a time series decomposition

First, I have loaded the computed time series into the tool and performed an initial empirical evaluation by plotting and measuring the graph shape. After performing some evaluation with different provided functions such as `msts` (multi-seasonal time series), I have noticed that the series has two different seasonality: one for the day, with period 24 and the other for the week, with period $24 \times 7 = 168$. In Figure 4.3, I have computed the decomposition of a time series, considering double seasonality [3].

In order to predict the traffic situation of a set period of time, I have extracted a sample of the time series and I have used it for training the model by using the Holt-Winters double seasonality function, called `dshw`. The function takes, as argument, the time series and the period of time to predict, and return a structure containing the forecasted values, in this case, the route transits for each hour [10].

I have also retrieved another sample, which is different and disjoint from the training one, for testing the accuracy of the model.

4.2.4 Experiments

In order to evaluate the accuracy of the created predictive model, I have performed different experiments. As mentioned before, for privacy reasons, I can't mention the real name of the roads or the routes direction.

I have considered three routes: the busiest routes in morning ($MT_{morning}$), evening ($MT_{evening}$) and in the whole day (MT_{total}). For each of the considered routes, I have performed the following experiments:

- **September** – Using the September 2018 dataset, I have retrieved the first three weeks as training sample and I have predicted the fourth one;
- **Summer** – Using both June and July 2019 datasets, I have used the whole June dataset as training sample and I have predicted the first week of July.

- **Unusual** – Using both September 2018 and May 2019 datasets, I have used the whole September dataset as training sample and I have predicted the first week of May.

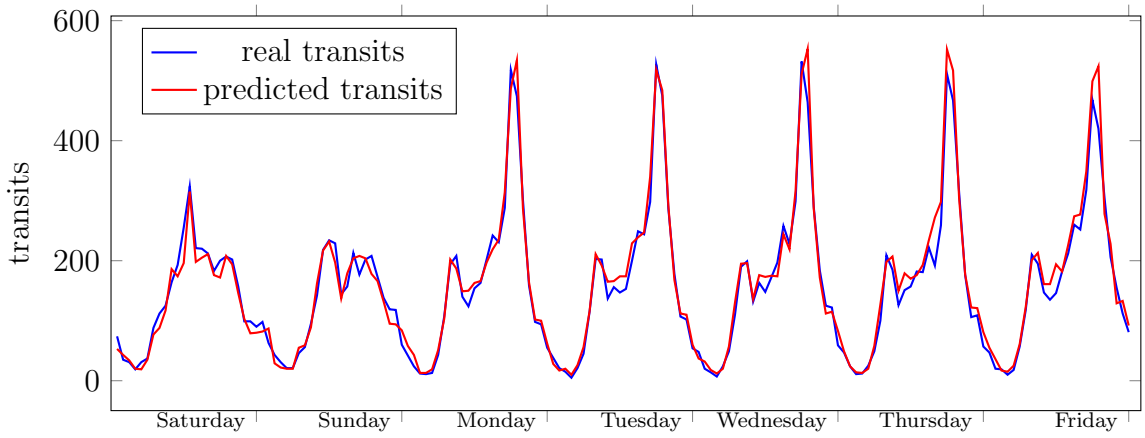
After computing the experiments, I have calculated, for each of them, the Mean Square Error (MSE) between the predicted and the real data. Results are shown in Table 4.6.

Routes	**MSE**		
	September	**Summer**	**Unusual**
$\boldsymbol{MT_{total}}$	462.56	575.12	1095.82
$\boldsymbol{MT_{morning}}$	379.32	579.04	899.31
$\boldsymbol{MT_{evening}}$	565.01	375.07	2204.45

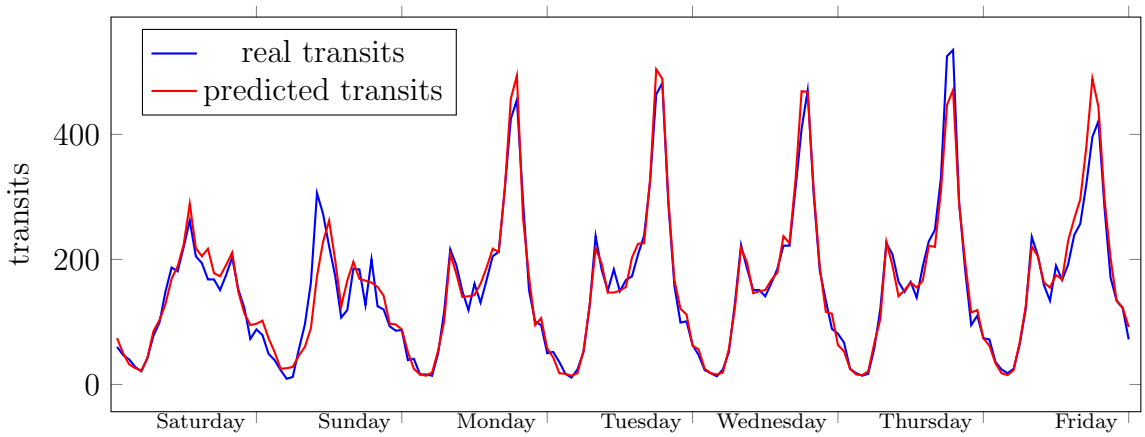
Table 4.6: Mean Squared Error of the time series experiments

I have also plotted the comparison of the two distributions for each experiment. It is very clear that the concentration of transits in workdays is different from the one in holidays. It's also visible the difference between each part of the day, considering the spikes in the early hours for the $MT_{morning}$ experiment shown in Figure 4.5 or the spikes in the late hours for the $MT_{evening}$ and MT_{total} experiments shown in both Figures 4.4 and4.6. Furthermore, considering the good *unusual* proof results, I have assumed that, probably, the behaviour of vehicles in Crema region is slightly predictable.

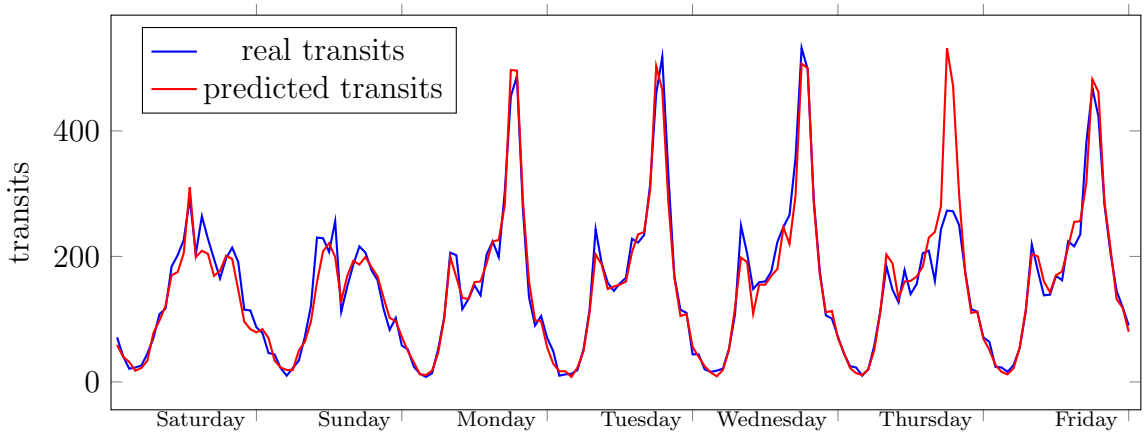
It is notable that for almost every proof, the predicted and real transits distributions are very similar, proving that the created model is very accurate.



(a) September

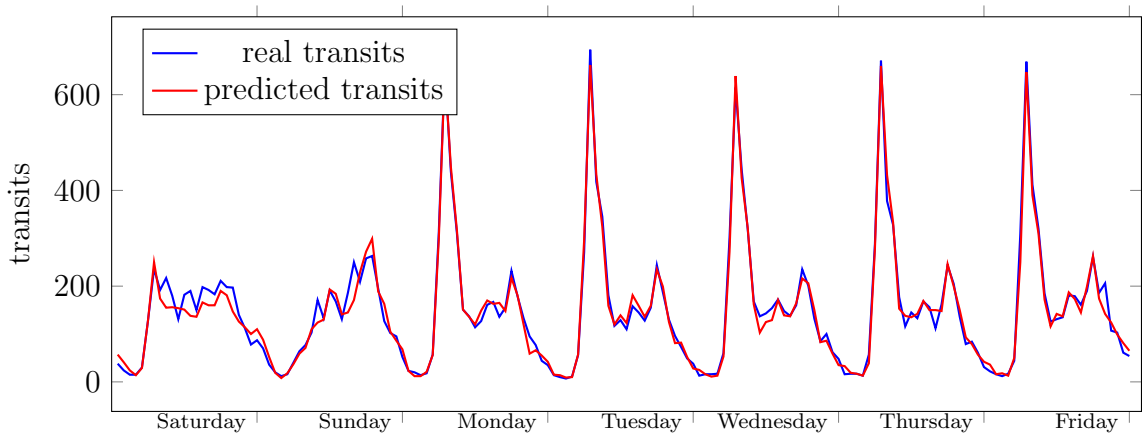


(b) Summer

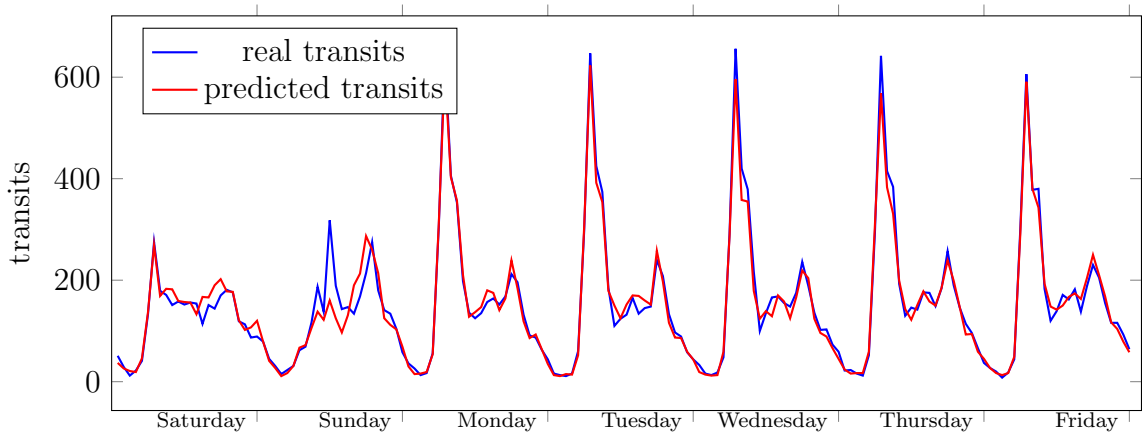


(c) Unusual

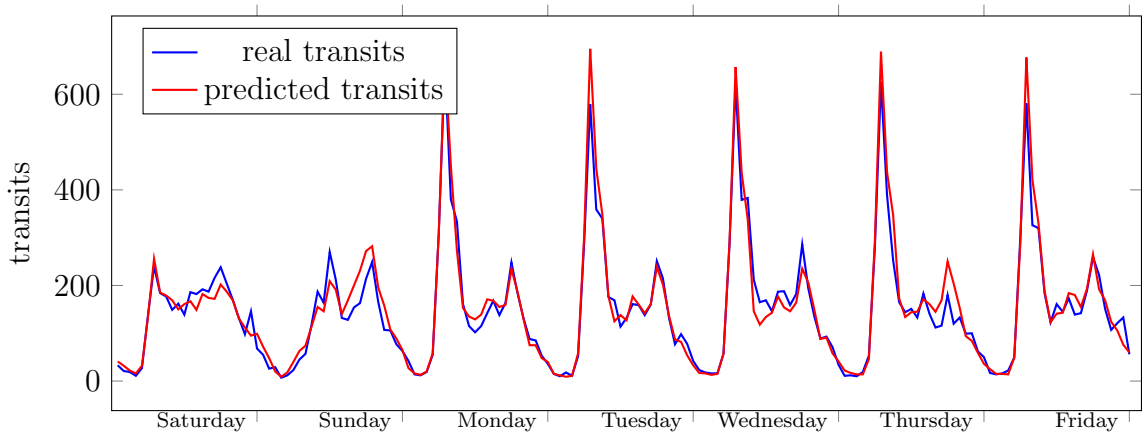
Figure 4.4: Predictions of MT_{total} route



(a) September

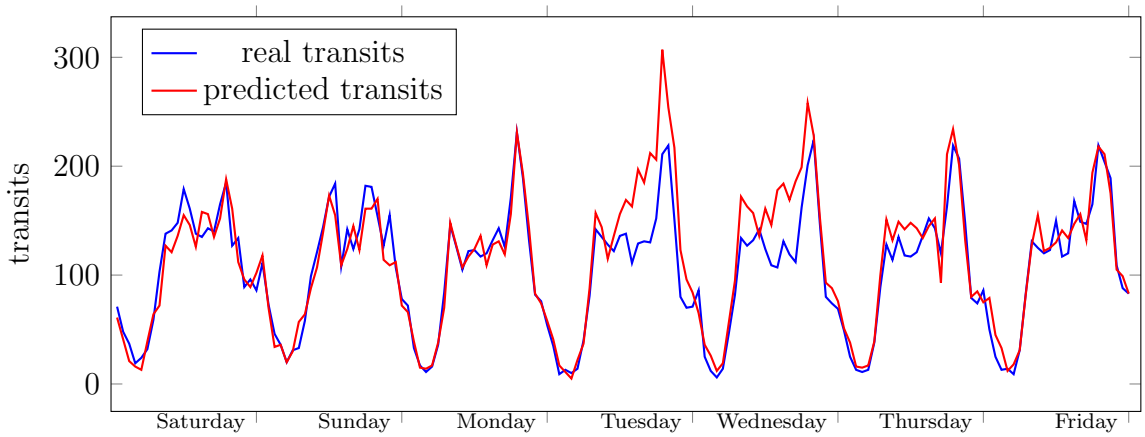


(b) Summer

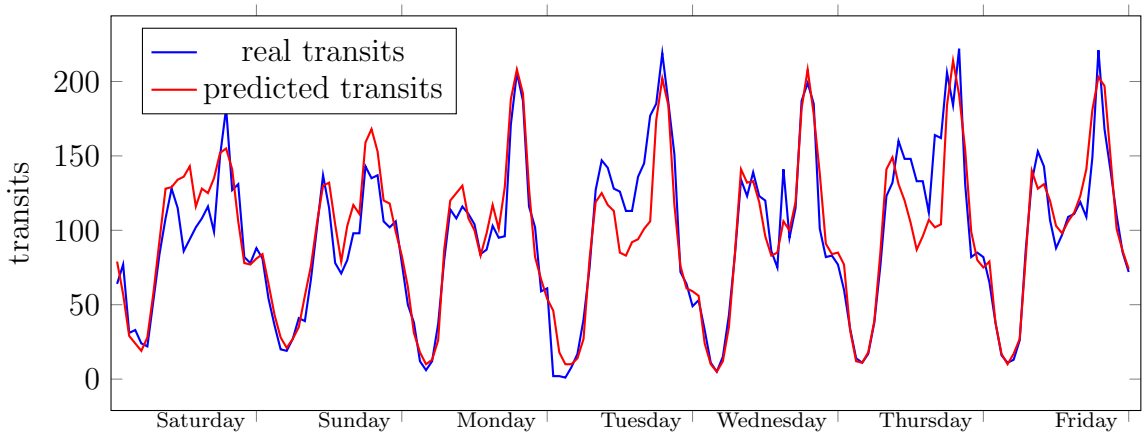


(c) Unusual

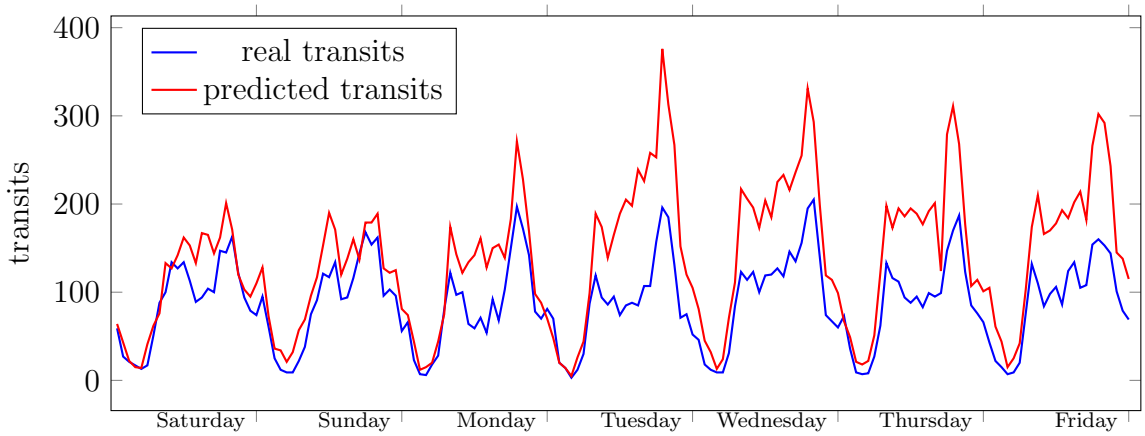
Figure 4.5: Predictions of $MT_{morning}$ route



(a) September



(b) Summer



(c) Unusual

Figure 4.6: Predictions of $MT_{evening}$ route

Chapter 5

Prescriptive Model

In this chapter I describe my prescriptive analysis on vehicles that trigger at least one alarm. As examples, alarms are triggered if the vehicle is stolen, with expired insurance, with expired inspection or marked as suspicious by the police department.

After extracting these entries from the dataset, I have compared two different approaches: an explicit integer programming model optimization, and the frequency based heuristic. The integer programming model determines the optimal position of a fixed number of patrols, in order to intercept the maximum number of vehicles producing alarms. The frequency based heuristic, instead, simply fixes patrols' positions near the gates with highest number of transits.

Finally, I have compared the two solutions in terms of overall expected number of vehicles producing alarms which are intercepted.

5.1 Paths extraction

Before applying the model, I have considered all the transits in the dataset which have set one of their `allarm_type_id` fields. Therefore, I have considered the datasets of May, June and July 2019 since September 2018 dataset is lacking alarms information. For simplicity of the model, I have not considered multiple alarms for the same vehicle.

I have considered each vehicle separately and I have extracted all their paths. Every vehicle is associated with a list of their paths and a list of counters that expresses how many time the considered vehicle has repeated their paths. A path is represented as an ordered list of cameras that have been monitored in a single day.

I have implemented this procedure as a Java application and I have applied the following steps: I have analysed the transits in pairs and, if they have the same plate and day, I add the new transit gate camera to the current vehicle's path. If the day is different, it means that the previous path has been completed and the current transit begins a new path for the same vehicle. If the plate is different, it means that the considered vehicle has no more transits and its creation is completed. It's added to the vehicles list and then, considering the new transit, I create a new vehicle and its first path.

After scanning the whole dataset, and filled the vehicles list, I have merged all the vehicles paths into one single list that contains also the paths frequencies.

The result are two correlated structures. The first is a binary matrix $a_{i,j}$, in which i denotes the path and j denotes the gate camera. The binary value of $a_{i,j}$ is true if the i path contains in its list the j gate camera, false otherwise. The second is a list w_i, which contains the frequency (or weight) for each path i.

The pseudo-code of this procedure is shown in Algorithm 6.

Algorithm 6 Paths extraction algorithm

```
                                                    ▷ Scanning the paths
previous ← NULL
for all line ∈ Dataset do
    if previous.vehicle_plate = line.vehicle_plate then
        if previous.date = line.date then
            path.add(line.cameraID)
        else
            vehicle.add(path)
            path ← newPath()
            path.add(line.cameraID)
        end if
    else
        list.add(vehicle)                           ▷ Changed vehicle
        vehicle ← newVehicle(line)
        path ← newPath()
    end if
end for
computePathProbabilities(list)      ▷ Applied only to the second method

result ← mergeVehiclePaths(list)
```

5.1.1 Another method

The method described in the previous section, however, in some cases, is slightly inaccurate. For example, if you consider a vehicle that has made 10 different paths and, from the model, the positioning of the patrols allows you to intercept all these routes, the result would consider the interception of 10 vehicles, while in reality it is a single vehicle.

To avoid this possible problem, I have implemented a second method that, instead of considering the count frequency of the path, compute the probability of taking this path compared to all the others of the actual vehicle.

Considering a vehicle's path frequency list f, the probability of taking each path is computed as follows:

$$p_i = \frac{f_i}{\sum\limits_{j \in paths} f_j}, \quad \forall i \in paths$$

By doing so, the resulted weight list w_i is calculated as a sum of probabilities, that correspond to the expected value of the intercepted vehicles if at least one patrol covers the i path [7].

The pseudo-code of this procedure is the same for both methods, expect for the path probabilities calculation, and it's described in Algorithm 6.

5.2 The model

The purpose of this model is to compare two different prescriptive approaches: an explicit integer programming model optimization and the frequency based heuristic.

5.2.1 Integer programming optimization

To build this model, I have implemented it in MathProg, using AMPL.[1].

After loading the paths and their weights, I have create the following model that aims to maximise the overall expected number of suspicious vehicles intercepted, by determining the optimal position of a fixed number of patrols.

The patrols are placed near gate cameras and their position is represented by a binary array y_j. The binary value y_j is 1 if one patrol has been placed near the position of gate camera j and 0 otherwise; also the total number patrols must not exceed k.

To compute the objective z, I have created a binary support variable x_i indicating if path i has been intercepted which means that at least one patrol is positioned next to one of the gate cameras present in path i.

$$\max z = \sum_{i \in paths} x_i \cdot w_i$$

$$s.t. \begin{cases} \sum_{j \in gateCams} y_j \leq k & \\ \sum_{j \in gateCams} a_{i,j} \cdot y_j \geq x_i, & \forall i \in paths \\ x_i \in \{0,1\}, & \forall i \in paths \\ y_j \in \{0,1\}, & \forall j \in gateCams \end{cases}$$

This model has been computed by the IBM Cplex [6].

5.2.2 Frequency based heuristic

To build this approach, I have created a Java Application. It determinates the overall expected number of suspicious vehicles intercepted by a fixed number of patrols.

In this case, the position of the patrols is already known and corresponds to the gate cameras that have monitored more transits during the evaluation period in descending order.

This approach tries to mirror one of the possible real behaviours of the police department.

5.3 Experiments

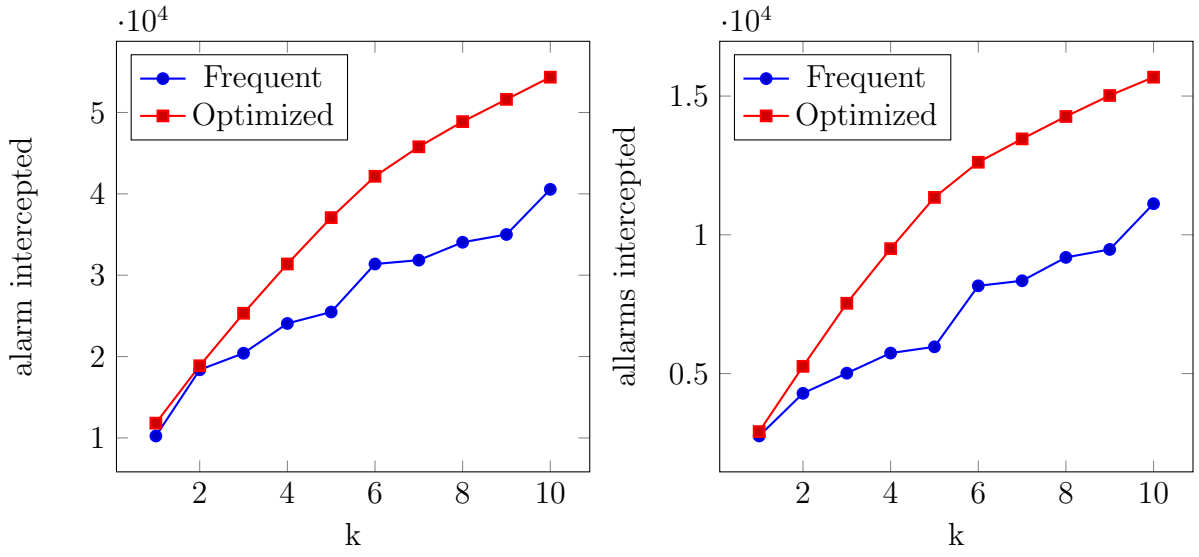
By fixing and computing different numbers of maximum patrols, I have compared the two solutions in terms of overall expected number of vehicles producing alarms which are intercepted. I have done three experiments:

- **AAT** – **A**ll **A**larm **T**ypes, considers every alarm regardless of the crime;
- **MFAT** – **M**ost **F**requent **A**larm **T**ype, considers the most frequent alarm type that has been registered on the dataset;
- **LFAT** – **L**ess **F**requent **A**larm **T**ypes, considers all alarm types, except the most frequent.

For privacy and security reasons, I can't mention the crime name corresponding to these values.

For every experiment, I have considered both path extraction methods, and for each of them, I have applied the two model approaches. I have chosen the range of the maximum number of patrols k which is from 1 to 10.

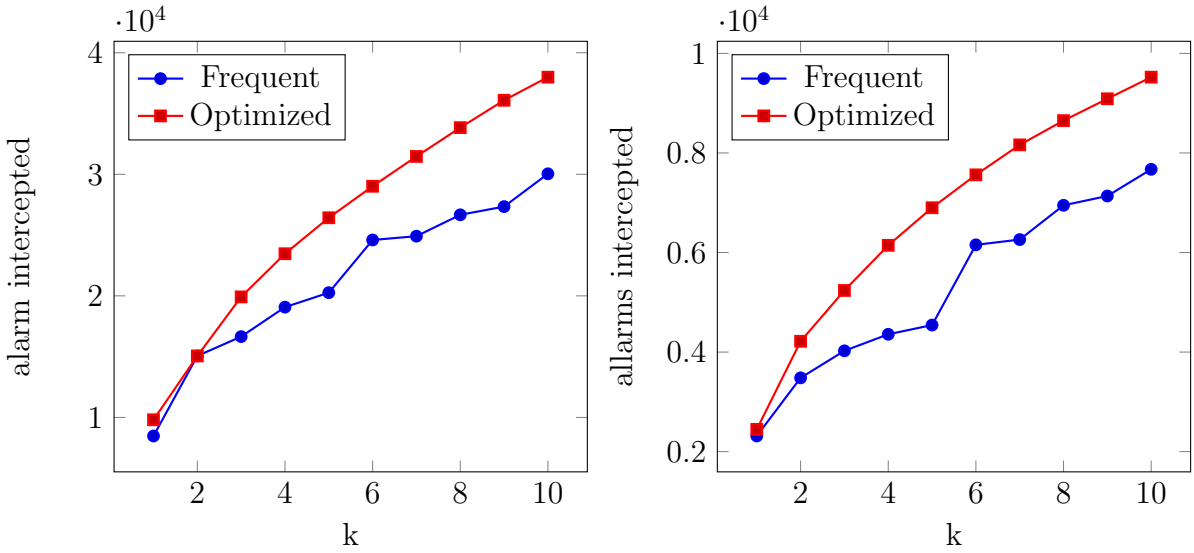
The graphical representation of the experiments are shown in Figures 5.1, 5.2 and 5.3. It can be seen from the graphs, in particular for experiments AAT and MFAT, that for a reduced number of patrols (less than 3), the two solutions are almost equivalent. For a greater number of patrols, the optimized approach continues to improve for all experiments.



(a) Counting method

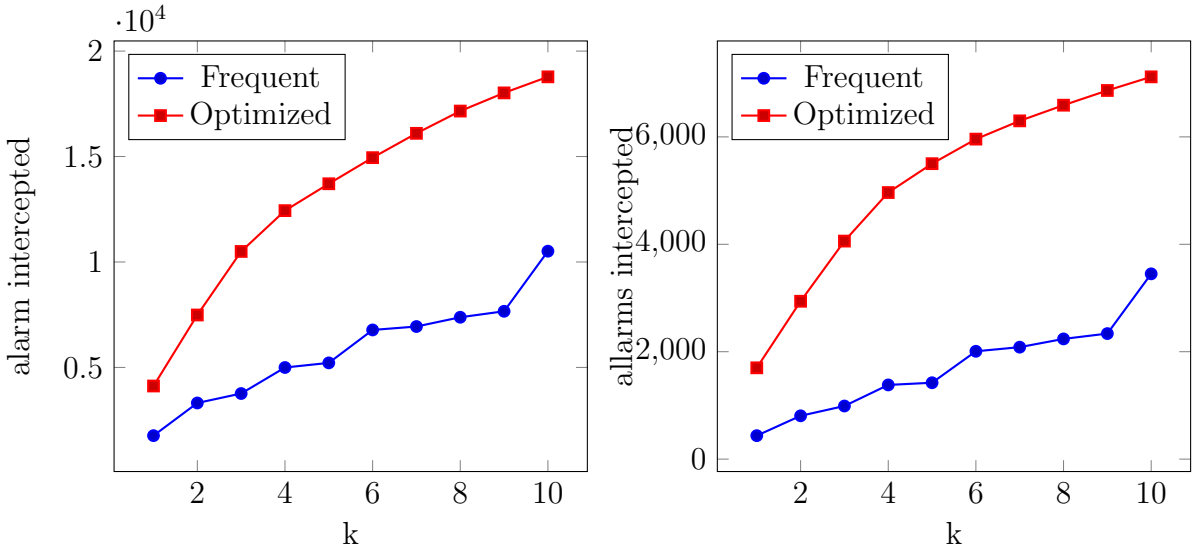
(b) Expected value method

Figure 5.1: Graphical representation of AAT



(a) Counting method (b) Expected value method

Figure 5.2: Graphical representation of MFAT



(a) Counting method (b) Expected value method

Figure 5.3: Graphical representation of LFAT

Chapter 6

Conclusions

The goals of this thesis is, after applying a preliminary analysis of the data, to create descriptive, predictive and prescriptive models and evaluate their accuracy by comparing them with real data.

For what concerns the *preliminary analysis*, I have found, in general, good quality in data, both in terms of completeness and accuracy. I have also built an effective visualization tool, in order to display vehicles' paths on a map.

For what concerns the *descriptive model*, I have aggregated different random variables, calculated on the real vehicles transits, to simulate the actual traffic situation. I have proved that the created model, in most cases, is very accurate and the number of transits are very similar to the real ones.

For what concerns the *predictive analysis*, I have created two model: in the first, I have analysed cluster tendency and actual gate cluster structures, and, in the second, I have performed time series analysis of traffic on links.

After evaluating the first model, the resulted clustering was fine for most of the considered timeslots. Unfortunately, I have also found that the applied clustering algorithm, considered only one of the parameters provided to perform the subdivision, making the results not always accurate.

After evaluating the second model, I have noticed that for most of the chosen links, the prediction was very accurate and, in some cases, almost identical. Based on these results, I have assumed that, probably, the behaviour of vehicles in Crema region is slightly predictable.

For what concerns the *prescriptive analysis*, I have considered all the transits of vehicles that trigger at least one alarm. I have compared two different approaches: an explicit integer programming model optimization, and the frequency based heuristic. The integer programming model determines the optimal position of a fixed number of patrols, in order to intercept the maximum number of vehicles producing alarms. The frequency based heuristic, instead, simply fixes patrols' positions near the gates with highest number of transits. For small numbers of patrols, the two solutions are very similar, but for higher number of patrols the optimal solution keeps improving.

6.1 Future implementations

Although the created models are very accurate, the prescriptive model could be improved by applying the two approaches to a training sample of the dataset, and evaluate them, using a disjoint testing sample, the overall expected number of vehicles producing alarms which are intercepted.

For clustering analysis, the model accuracy may increase if more relevant features are determined from the gate cameras or if it is possible to decide, a priori, the representatives of different class of possible clusters.

Bibliography

[1] Robert Fourer, David Gay, and Brian Kernighan. *AMPL: A Modeling Language for Mathematical Programming*, volume 36. 01 2002.

[2] Gianpaolo Ghiani, Gilbert Laporte, and Roberto Musmanno. *Introduction to Logistics Systems Planning and Control (Wiley Interscience Series in Systems and Optimization)*. John Wiley; Sons, Inc., USA, 2004.

[3] Phillip G. Gould, Anne B. Koehler, J. Keith Ord, Ralph D. Snyder, Rob J. Hyndman, and Farshid Vahid-Araghi. Forecasting time series with multiple seasonal patterns. *European Journal of Operational Research*, 191(1):207 – 222, 2008.

[4] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[5] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[6] IBM Office. *CPLEX User MANUAL*. IBM, 12th edition, 2018.

[7] Sheldon M. Ross. *Simulation, Fourth Edition*. Academic Press, Inc., Orlando, FL, USA, 2006.

[8] Consorzio.IT S.p.A. Progetto varchi. `https://www.consorzioit.net/servizi/ProgettoVarchi`. Accessed: 2019-10-25.

[9] Springler. *The Concise Encyclopedia of Statistics, Bernoulli Distribution*. Springer New York, New York, NY, 2008.

[10] James Taylor. Short-term electricity demand forecasting using double seasonal exponential smoothing. *Journal of Operational Research Society*, 54:799–805, 08 2003.

List of Algorithms

List of Figures

List of Tables

Ringraziamenti

Vorrei ringraziare, innanzitutto, il professor *Alberto Ceselli* per essere sempre stato disponibile in questi anni e per avermi aiutato e guidato nella stesura di questo lavoro.

Ringrazio il *Consorzio.IT* ed, in particolare, il dottor *Andrea Tironi* per aver fornito il materiale necessario.

Ringrazio i miei genitori, *Roberta* e *Luigi* per avermi sempre sostenuto ed incoraggiato in questi 5 anni di percorso universitario.

Ringrazio mio fratello *Nicola* e mia cognata *Simona* per essermi sempre stati vicino.

Ringrazio i miei nonni materni, *Lucia* e *Francesco* ed i miei nonni paterni, *Maria* e *Antonio* per aver sempre creduto in me.

Un ringraziamento speciale va a *Luca Diedolo*, per essere sempre stato un grande amico e compagno di studi, e per aver svolto insieme parte di questo progetto.

Ringrazio tutte le persone che ho conosciuto in università, per essere state più che semplici colleghi, in particolare *William*, *Sara* e *Ilaria*.

Ringrazio inoltre tutti gli insegnanti ed i collaboratori universitari che durante questo percorso sono stati di grande aiuto.

Ringrazio i miei amici di sempre *Sunny*, *Luca D.* e *Luca F.* che hanno condiviso con me gioie, sacrifici e successi.

Ringrazio infine tutti i miei familiari e tutte le persone che in questi anni mi sono stati vicini.

Grazie a tutti!