POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

ASSOCIATIVE CLASSIFICATION OF SPATIO-TEMPORAL DATA

Supervisors

Prof. Paolo GARZA

Dott. Luca COLOMBA

Candidate

Salvatore Stefano FURNARI

April 2023

Summary

Among the data mining tasks, the extraction of patterns that show relevant spatial and temporal dependencies among data is one of the most useful in order to deal with a wide range of fields of application. In particular, the sector of the *digital platforms* is a really huge source of this type of patterns, since the services that these digital applications are offering are distributed in time and space.
An example of this can be a food delivery service, which keeps track of the historical data about the orders submitted by the users (with details about the exact times of the orderings and of the trips for the deliveries) and the associated spatial information such as the location of the users and restaurants and the position of the riders. All these information can be put together and exploited in order for the algorithm behind the digital application to match user's orders and riders in the most efficient way possible.

In this thesis, we take into account the analysis of years of historical data about a bike-sharing service based in San Francisco. The records contained into the dataset are structured as a minute by minute report of the bike stations status, meaning the number of bikes currently parked there and the number of docks currently available. These data can be a good source of information for the fleet manager, since they can help them understand which kind of interventions have to be done in order to always have the appropriate amount of bikes in the various stations around the cities.
The main point to be defined is how to transform the data in order to discover patterns which embody both the spatial and temporal dimensions, in a way that is not specific of certain trajectories observed over a region of interest: so we propose an efficient algorithm to extract this kind of patterns and validate its efficiency and effectiveness on real data. To achieve this goal, we aim to generalize this type of information by detecting sequences of *events* of interest, reporting *spatiotemporally invariant* properties. In this way, the data gain more informative power, since we reduce the huge amount of timestamp records to sequential data of the typology *event_station_time window* thanks to a *discretization* technique that revolves around a *reference* (with respect to the *stations* and the *time*), where:

- **Event**: is the *discretization* of the variety of status that a *station* can have in terms of occupancy. In particular, we have defined a set of critical conditions represented by:
  - **Full**, which represents the situation when the number of docks available is equal to 0;
  - **Almost full**, which represents the situation when the number of docks available is less or equal than a certain threshold;
  - **Empty**, which represents the situation when the number of bikes available is equal to 0;
  - **Increase**, which represents the situation when the number of bikes available is greater than the one of the previous timestamp;
  - **Decrease**, which represents the situation when the number of bikes available is less than the one of the previous timestamp.
- **Station**: to represent the *stations* belonging to the bike-sharing service, the algorithm makes use of the *discretization* of the distance with respect to the *reference station*. Each sequence of *events* takes into account the current and past status of a specific neighborhood of the *reference station*, composed as follows (depending on the setting of the algorithm which extracts the *spatiotemporally invariant* patterns):
  - **Radius**, meaning that the neighborhood of the *reference station* will consist of all the stations located within the **Spatial_steps** that have been set.
    For example, if this parameter is set to 5, a *sequence* will report all the *events* belonging to the **Event_types** set that have happened in all the stations placed within a circle of radius **Spatial_threshold** $*$ **Spatial_steps**.
  - **Incoming**, meaning that the neighborhood of the *reference station* will consist of the **Top_n** stations which have the highest number of records into the *trip* dataset (i.e. records where the **Start_station_id** is the neighbor, while the **End_station_id** is the *reference station*). Doing so, we are selecting the stations from which the highest number of bikes is arriving to the *reference station*, regardless of their distance with respect to the *reference station*.
- **Time**: to represent the *time* at which the *events* take place, we do not report the specific details such as the date and the hour. The algorithm makes use of the *discretization* of the timeline subdividing it in *time windows* of a

certain length: the trigger *event* that happens in the *reference station* defines the *reference window* (i.e. the so called "instant zero"); then, the algorithm generates sequences of *events* which take into account the current and past status of a specific neighborhood of the *reference station*, looking at a certain number of *time windows* (i.e. the duration of the *time horizon* under analysis, depending on the setting of the algorithm which extracts the *spatiotemporally invariant* patterns).

The extraction of these *spatiotemporally invariant* patterns is useful for us to deploy an *associative classifier*. Indeed, we make use of an algorithm for *sequential data mining* (i.e. **Prefixspan**) in order to detect these type of patterns from the sequences of correlated *events* of interest (which are the result of the data transformation of the original *status* dataset, necessary to exploit this classification technique). The steps we take to adopt this classifier are the following:

- Application of the **Prefixspan** algorithm to get the set of *association rules*;
- Selection of the rules which contain at least one *event* associated to the *reference station* and happened at the "instant zero";
- Filtering out the rule list according to the critical condition associated to the *reference station* and happened at the "instant zero" that we are interested to classify (keep just the rules which contain it, and not having *events* of interest happened in different *stations*);
- Setting of a minimum *confidence* threshold to select the rules that have a certain "strength";
- Setting of a minimum *support* threshold in order to keep just a certain number of rules (setting so an order of magnitude for the adopted rules).

Once we have selected the desired number of *association rules*, we perform a binary classification task that involves the recognition of some sort of critical condition for the bike stations such as the lack of bikes or the complete occupancy of the docks. The *associative classifier* looks for a correspondence between a certain amount of rules (defined by a previously set *matching threshold*) and the sequence for which we want to predict its class label. In other words, a *sequence* is classified as belonging to the *positive* class if it contains a number of selected rules greater or equal than the minimum *matching* threshold (so that they are *subsets* of it, except for the "instant zero", which is the timeslot to be predicted and so we do not look for pattern matching in that period). Instead, a *sequence* actually belongs to the *positive* class if it contains the critical condition of interest associated to the *reference station* at the "instant zero".

We compare these results with other models: a **Baseline** which consists of an *associative classifier* that makes use of just a single simple and intuitive rule, and classical algorithms such as **Decision Tree** and **Random Forest**.
To deploy them, we need tabular data with couples attribute-value. Our choice is to get them from the already generated *sequences* from the *status* dataset (the ones used to extract the *association rules*). All the columns are of *binary* type, signalling if the situation represented by it has occurred. So, the data representation will be different depending if the neighborhood of the *reference station* is built in the *radius* or the *incoming* mode:

- **Radius**: the input data for the *classification* algorithms have a column for each combination of *event*, *station* and *time window* as set by the parameters **Event_types**, **Spatial_steps** and **Temporal_steps** of the *sequence generation* algorithm (except for the "instant zero", for which we just report the class label, since we do not want to use features belonging to the same *time window* of the class label for making predictions);

- **Incoming**: the input data for the *classification* algorithms have a column for each combination of *event* and *time window* related to the *reference station* as set by the parameters **Event_types** and **Temporal_steps** of the *sequence generation* algorithm (except for the *decrease event* in case the *positive* class is associated with the *full* condition), and a column for each *time window* related to the whole neighborhood (reporting if there is at least one *station* belonging to the neighborhood of the *reference station* which has been in the *decrease* condition during that specific *time window*). Again, for the "instant zero", we just report the class label, since we do not want to use features belonging to the same *time window* of the class label for making predictions;

For some experiments, we do not keep just the whole selection of data, but instead we subdivide the dataset partitions into timeslots of 4 hours (00-03, 04-07, 08-11, 12-15, 16-19, 20-23), in order to better assess the models performances in the various parts of the day and their different trends. Indeed, each timeslot is characterized by its own type of traffic, determined by the necessity of people to go to work, university, gym... leaving inevitably some timeslots with more stability in the stations' status, while the others have a more frenetic pace of changes.

As we see performing the experiments, it is difficult to conciliate satisfactory values of the metrics *precision* and *recall* for the minority class. With a poor *precision*, we end up having a lot of *false positives*: in our case study, that implies assuming that a station has filled all its docks, and so it is ready to be deprived of some bikes to be destined to other stations that need them, even if actually is not. With a poor *recall*, we end up having a lot of *false negatives*: in our case study, that implies assuming that a station has not filled all its docks, and so it is ready

to host incoming bikes, even if actually is not. Determining which metric has to be optimized is a choice that can be made only by means of an accurate analysis by the decision-makers, whose can numerically evaluate the cost and benefits of these scenarios and identify the most suitable option, even with differentiated solutions according to the geographical location of the single stations.
What we observe in general is that the traditional classifiers perform better for what concerns the *precision* (in particular the *random forest*), while the *associative classifier* for what concerns the *recall*.

Acknowledgements

To my family, that has supported me even during the toughest moments.

To my friends, that have filled my academic path with wonderful memories.

To the discography of Death, my soundtrack during the writing of this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Data-driven decision-making processes are being increasingly adopted in many business and social fields, such as industrial production and healthcare. This trend has emerged thanks to the availability of huge volumes of application-specific data, which allow for a proper training procedure of *machine learning* algorithms. Being exposed to this amount of examples, they can infer patterns that are hard to detect for human operators, showing new perspectives to the decision-makers.

This thesis explores the promising possibilities given by *machine learning*, using as a case study a bike-sharing service which operates in the San Francisco Bay Area (Bay Area Bike Share). The task at hand is specifically a **classification of spatio-temporal data**, since we aim to predict labels for samples which present both spatial (localization of bike stations) and temporal (date and time of bike stations' status) details. In particular, labels are events of interest regarding stations' occupancy: indeed, the objective is to provide insights useful for the bike-sharing service management, such as setting the most appropriate frequency of bike supply for the stations.
As for the deployed classification techniques, we compare more traditional algorithms (Decision Tree and Random Forest) with our implementation of an **associative classifier**, which looks for matchings between spatio-temporal patterns that happened in the past and similar situations in the future. A distinctive feature of our classifier is that the extracted patterns are not based on absolute references to stations' locations and time, but instead on *relative* ones, making it a **spatio-temporally invariant** algorithm.

Analyses are conducted considering both the whole day and dataset partitions by time slots; moreover, we take into account different thresholds to consider successful matchings between samples to be classified and patterns extracted from the historical data.

The remaining of the thesis is organized as follows. In Chapter 2, we revise some bibliography useful to introduce the deployed *classification algorithms* and the theoretical concepts at the basis of them, such as the notion of *association rules* and the most adopted algorithms to extract frequent patterns (**Apriori**, **FP-tree** and **Prefixspan**). In Chapter 3, we delve into the dataset structure, the splitting strategy in order to design a proper training pipeline for the classification models and the implemented algorithm to transform the original *status* dataset into a *sequential* one (in order to extract *spatio-temporal* patterns). In Chapter 4, we introduce the different experimental settings, analyze and compare the different classification model. In Chapter 5, we draw our conclusions about the obtained results and we indicate some possible directions to take for further research and improvement.

Chapter 2

Related work

In this thesis, we deploy three different classifiers in order to detect the critical conditions for the *reference station*, that are described in more detail in the subsequent paragraphs.

The main concepts related to data mining and machine learning which are introduced in the following paragraphs can be found in [1] and [2] respectively.

2.1 Associative classifier

This classifier makes use of the *association rules* extracted from a *transactional* database in order to perform predictions.
This typology of database contains as entry a *transaction*, which can be defined as a set of *items* where their order does not count. For example, we can consider *market basket* data, in which a *transaction* is the purchase of a customer, whereas the *items* are the products that have been bought, as we can see in the Figure 2.1.

TID	Items
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diapers, Milk
4	Beer, Bread, Diapers, Milk
5	Coke, Diapers, Milk
...	...

Figure 2.1: Example of a *transactional* database: tickets at a supermarket counter.

2.1.1 Association rules

The objective of *association rule mining* is to extract frequent patterns from a *transactional database*, which highlights some kind of correlation between some *items*. An example with the already shown database could be:

$$coke, diapers \implies milk \tag{2.1}$$

The *items* placed before the arrow constitute the rule *body*, whereas those after constitute the rule *head*. The arrow stands for a *co-occurrence* of the *items* belonging to the rule, and not for a direct *causality*. This means that it is not always true that a transaction containing *coke* and *diapers* will also contain *milk*, but that this correlation is verified with a certain probability.

In order to extract *association rules* with certain "strength" and "quality", we first have to filter out the *transactional database* with a preprocessing algorithm. In particular, we want that our *association rules* have not as *body* and *head* whatever kind of *items*, but instead that together they form a *frequent itemset*, meaning that the *support* (i.e. the fraction of *transactions* that contains an *itemset*) is greater or equal than a certain threshold. With the rule example made before, we can define

the *support* of a rule as the *support* of the *itemset* containing both the *body* and the *head*:

$$support = \frac{\#\{coke, diapers, milk\}}{|T|} \tag{2.2}$$

where on the numerator we have the cardinality of the rule, while on the denominator the cardinality of the entire database. This quantity can also be viewed as an estimation of the probability of the rule inside the database.
Another evaluation metric for which we can set a minimum threshold for the extracted *association rules* is the *confidence*, which is the frequency of occurrence of the *head* inside the *transactions* that contain the *body*. Put into formula:

$$confidence = \frac{sup(coke, diapers, milk)}{sup(coke, diapers)} \tag{2.3}$$

where on the numerator we have the *support* of the rule, while on the denominator the *support* of the *body*. This quantity can also be viewed as an estimation of the conditional probability of having the *head* in a rule that contains the *body*, and so it represents the "strength" of the "implication" symbol contained in the rule.
So, *association rule mining* will lead to a result that is *complete* (i.e. all the rules satisfy both the thresholds) and *correct* (i.e. only the rules which satisfy both the thresholds).

For computational constraints, the adoption of a *brute-force* approach is usually avoided: create each possible *association rule* by permutating all the *items* inside the *transactional database*, and then prune all the rules that do not respect the threshold requirements.
Indeed, given a database with a number d of *items*, we end up with 2^d *itemsets*, as shown in the Figure 2.2.

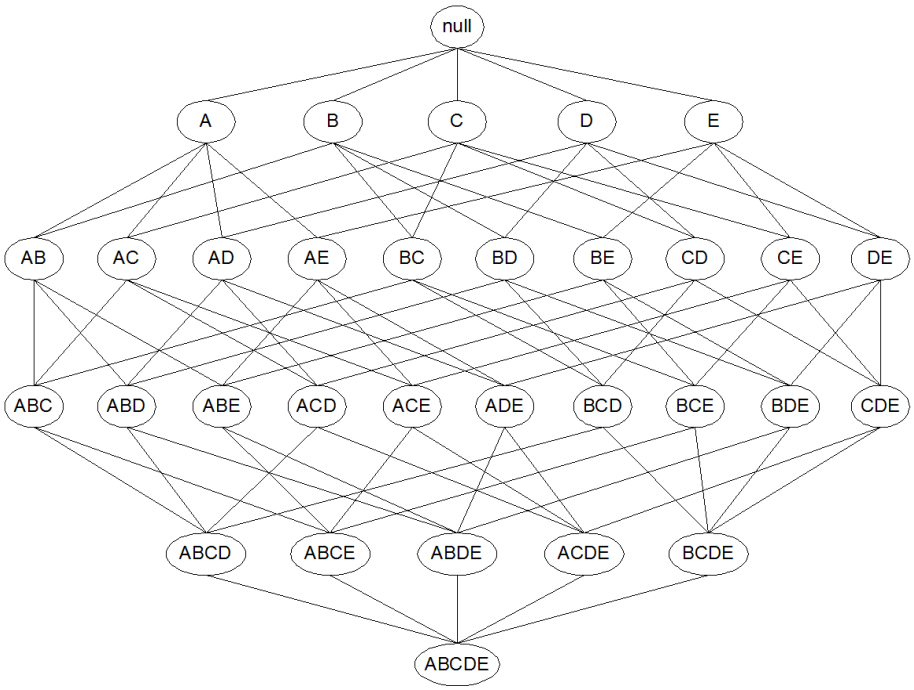

Figure 2.2: Example of the *lattice* drawn from a *transactional database* with five *items*.

So, for each candidate *itemset* contained in the *lattice*, we should scan the whole database in order to calculate its *support*, leading to a computational complexity equal to $O(|T|2^d w)$, where w is the maximum length of a *transaction*.

Instead, the method of firstly generating the *frequent itemsets* and then creating the rules from them is preferred (with all the possible combinations of *body* and *head*, so all possible binary partitioning of each *frequent itemset*, usually setting a threshold for the *confidence*). Most adopted algorithms for the generation of *frequent itemsets* are **Apriori** and **FPgrowth**.

Apriori

The first one is based on the *Apriori principle* built upon the *antimonotone property* of the *support* metric which is, given two *itemsets* A and B:

$$A \subseteq B \implies sup(A) \geq sup(B) \tag{2.4}$$

This means that each *itemset* contained entirely into another one has always a greater or equal *support*: so, if we find a certain *itemset* which does not respect the minimum *support* threshold, then all its *supersets* will behave in the same way,

and so can be pruned from the *lattice* of the candidate *frequent itemsets*, as shown in the Figure 2.3.

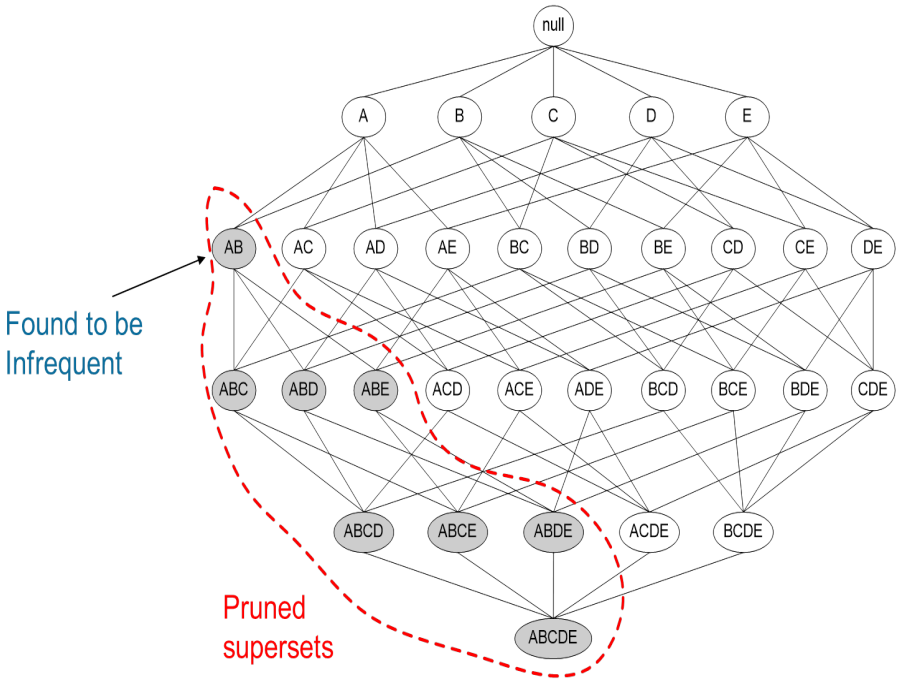

Figure 2.3: Application of the *Apriori principle*.

This algorithm adopts a *level-based* approach, since step by step it extracts the *frequent itemsets* of increasing length, starting from the previously extracted ones of length k. The steps for each iteration are as follows:

1. **Candidate generation**: it is the step that allows to find the *itemsets* which can be potentially frequent. It consists of the following:

   - **Join step**: the candidate *itemsets* of length $k+1$ are generated by combining between each other the *frequent itemsets* of length k.
     This can be done for example by sorting the *frequent itemsets* of length k in lexicographical order, and then joining each one of them with every other *itemset* which shares the same prefix of length $k-1$.
   - **Prune step**: the algorithm applies the *Apriori principle*, so it discards the *itemsets* of length $k+1$ which contain at least one *itemset* of length k that is *not* frequent.

2. **Frequent itemset generation**: it is the step that determines, given the potentially frequent *itemsets* from the previous step, which are the actually frequent *itemsets* of length $k+1$. It consists of the following:

- **Scan step**: the *transactional database* is scanned to calculate the *support* of the selected candidate *itemsets* of length *k+1*.
- **Prune step**: the candidate *itemsets* of length *k+1* which have a *support* below the minimum threshold are discarded.

However, this algorithm still has some performance issues, since the *frequent itemsets* are found increasing step by step their length, and so the extraction of the *frequent itemsets* of high length requires before the generation of all the frequent subsets. Indeed, especially the generation of the candidate *itemsets* of length 2 could be very computationally expensive, given the fact that they are more probable to be frequent. Moreover, the computational complexity of the *support* calculation for the candidate *itemsets* increases linearly with the length of the longest *frequent itemset*. So, the factors affecting performance can be listed as follows:

- **Minimum support threshold**: if it is too low, we risk to have a too large number of candidates and an increasing length of *frequent itemsets*, which affects the computational complexity of the *support* calculation;
- **Number of distinct items inside the dataset**: this increases the memory space needed to store the count of the *support* and the computational costs for the extraction of *frequent itemsets*;
- **Size of the transactional database**: it determines how long it takes for the algorithm to scan through the database in order to calculate the *support* of the candidate *itemsets*;
- **Average width of the transactions**: if it is high, we may have *frequent itemsets* of greater length, and so a much longer runtime for the algorithm.

FP-tree

This issues can be addressed by the **FPgrowth** algorithm, which represents the *transactional database* in a compressed way building an *FP-tree*, adopting a *recursive* approach for the *frequent itemset mining* which decomposes it into smaller subtasks. By doing so, the algorithm just has to perform two scans of the whole *transactional database*, to calculate the *support* of the *items* and to build the *FP-tree*. The steps are as follows:

1. Collect all the *items* in the *transactional database* which have a *support* greater or equal than the minimum threshold;
2. Build a new database (i.e. the *header table*), which is a list of the selected *items* sorted in decreasing order of *support*;

3. Scan the *transactional database* in order to build the tree. So, for each *transaction* inside the database, we place the *items* in the same order as the *header table* and then we insert it in the tree (using an existing path if it has a common prefix, or creating a new branch otherwise);

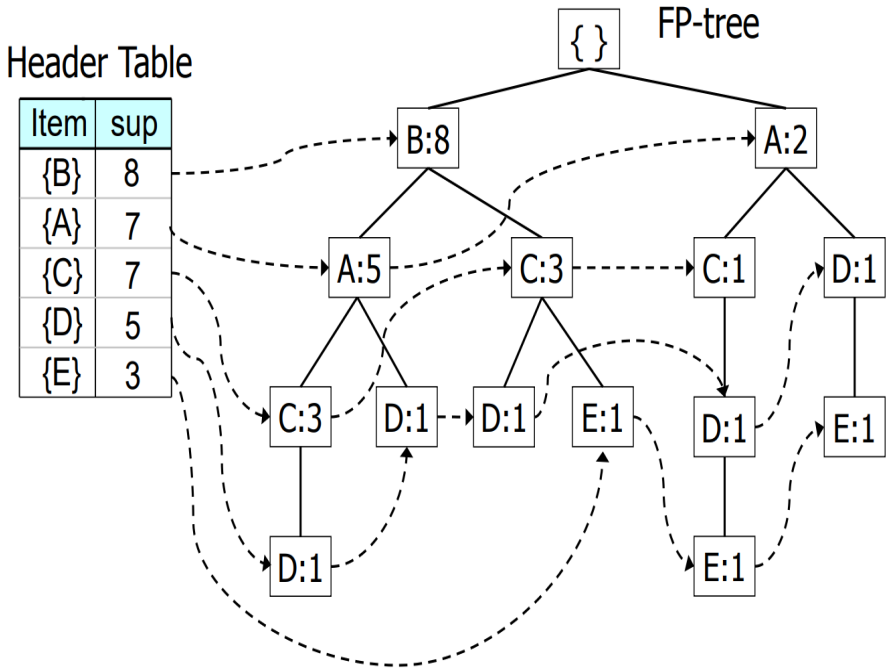

Figure 2.4: Example of a *FP-tree*.

As we can see in Figure 2.4, the number inside each node of the tree represents the *support* of the *item* inside certain *transactions*. For example, looking at the leftmost path, there are:

- Eight *transactions* with the *item* B;
- Five *transactions* with the *itemset* BA;
- Three *transactions* with the *itemset* BAC;
- One *transaction* with the *itemset* BACD;

At this point, the algorithm begins to scan the *header table* from the bottom, extracting the *frequent itemsets* which contain the *item* under analysis and all the other ones that are above it in the *header table*, by means of *recursive* invocation of the algorithm on the *conditional pattern base* for the *item* under analysis. This is a reduced version of the original *transactional database* which contains all the *transactions* which are prefixes of the considered *item*, with a *support* equal to

the one of the considered *item* inside the tree paths. So, the algorithm builds the *header table* conditioned on the analyzed *item*, and then again scans it from the bottom to then recursively apply the algorithm on the *conditional pattern base* with respect to the two *items* under analysis, and so on.
The extracted *frequent itemsets* will have the *support* associated with the last *item* in the set, as reported in the *header table*. The *recursion* goes forward until we end up with a *conditional pattern base* which has no *frequent items*: in this case the algorithm goes back to the previous *header table*, scanning it in the upward direction.

What emerges is that the choice of the threshold for the minimum *support* is not trivial. If a too high threshold is set, we risk discarding *itemsets* which contain *items* that have for their intrinsic nature a rare frequency into the *transactional database* but are anyway interesting. Taking again the example in which a *transaction* consists of the *items* purchased by a customer, expensive products such as pieces of jewelry could take this role. On the other hand, if the threshold is too low, the extraction of the *association rules* could become very expensive as the number of *frequent itemsets* becomes huge.

2.1.2 Classification by pattern-matching

Association rules can be used for the *machine learning* task of *classification*. Indeed, a classical structured database can be viewed as *transactional* if we consider as *items* the couples attribute-value, as well as the class label, which takes the role of the *head* of the *association rule*. So, we can proceed to extract the rules from the database, as in the example reported in Figure 2.5.

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
human	warm	yes	no	no	mammals
python	cold	no	no	no	reptiles
salmon	cold	no	no	yes	fishes
whale	warm	yes	no	yes	mammals
frog	cold	no	no	sometimes	amphibians
komodo	cold	no	no	no	reptiles
bat	warm	yes	yes	no	mammals
pigeon	warm	no	yes	no	birds
cat	warm	yes	no	no	mammals
leopard shark	cold	yes	no	yes	fishes
turtle	cold	no	no	sometimes	reptiles
penguin	warm	no	no	sometimes	birds
porcupine	warm	yes	no	no	mammals
eel	cold	no	no	yes	fishes
salamander	cold	no	no	sometimes	amphibians
gila monster	cold	no	no	no	reptiles
platypus	warm	no	no	no	mammals
owl	warm	no	yes	no	birds
dolphin	warm	yes	no	yes	mammals
eagle	warm	no	yes	no	birds

R1: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds
R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes
R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals
R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles
R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

Figure 2.5: Example of rules extracted from table rows.

Having done this, we then check if the instance to be classified is *covered* by any of the extracted rules, meaning that the attributes of the instance satisfy the *body* of the rule, as we see in Figure 2.6.

R1: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds
R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes
R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals
R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles
R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
hawk	warm	no	yes	no	?
grizzly bear	warm	yes	no	no	?

Rule R1 covers a hawk => Bird
Rule R3 covers the grizzly bear => Mammal

Figure 2.6: Example of rule-based classification.

A single instance can be covered by more than one rule, so they have to be sorted with respect to their *support* and *confidence* measures in order to define a priority among rules, and assign the class which is associated to the *head* of the highest

ranked rule that *covers* it. Moreover, we have to define a default class which is assigned in case the instance is not *covered* by any rule.

Advantages of this approach is that we are building an *interpretable* model, given the fact that we can inspect the selected rules to understand the patterns involved in the prediction of a certain class. The process of classification is pretty efficient and still possible even in presence of *missing data* (the algorithm simply skips those rules for the *coverage* check); also, we have good scalability in terms of the training set size.
For what concerns the **drawbacks**, we have that the rule extraction could be computationally expensive (depending on the threshold of minimum *support*) and that the scalability is not so good in terms of number and domain of the attributes, since we could end up with a massive amount of *itemsets* to be checked.

The dataset analyzed in this thesis is not suitable for an *associative classifier*, since it is a *logfile* which reports the status of each station in terms of bikes and docks available almost minute by minute over two years of data. This makes it unfeasible to use as *items* the couples attribute-value, since the domain of the attributes is too wide, especially the one of the time; moreover, we have to explicitly define a label to be classified.
We have chosen to preprocess the data adopting the technique of *discretization*: the informations about the bikes and docks available become a discretized set of *events*, the *station_id* becomes the discretized distance with respect to a *reference station* and the timestamp becomes a *time window* discretized with respect to a reference one inside a predefined time horizon. In this way, any row of the original dataset simply becomes an *item* identified by the union of an *event*, a *station* and a *time window*; a *transaction* instead will be the aggregation of many rows inside the chosen time horizon.

2.1.3 Prefixspan

So, we have to use an algorithm which is capable of extracting *association rules* from such a database. Actually, to achieve this goal, we rely on an algorithm which performs *sequence mining* to extract the spatio-temporal patterns of our interest, and then we interpret them as *association rules*.
Since we want to predict a situation of criticality for a station based on the conditions of its neighborhood during a predefined time horizon in the past, we identify as *body* of the rule the set of events temporally localized in the past *time windows*, while the *head* of the rule is identified as the set of events temporally localized in the last *time window* (i.e. the *time window* for which we want to predict a critical condition for a certain station).

For this purpose, we adopt **Prefixspan** [3], which stands for *prefix-projected sequential pattern mining*. This algorithm has been developed specifically to address the task of *sequential pattern mining*, which aims to extract frequent subsequences as patterns in a *sequence database*. This is the case of a wide range of applications, such as *customer purchase behavior*, *Web access patterns*, *DNA sequences analysis* and, as in our case, analysis of patterns distributed in time and space. The definition of this task has been given first in [4]: *Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items, and given a user-specified min_support threshold, sequential pattern mining is to find all of the frequent subsequences, i.e., the subsequences whose occurrence frequency in the set of sequences is no less than min_support.*
Taking as example our dataset (as explained in more details in Chapter 3), the *elements* of a *sequence* are the *time windows* contained in the analyzed time horizon (so reported in chronological order), while the *items* inside it are the triplets *event_station_time window* (considered by the algorithm in alphabetical order).
A *sequence* is a *subsequence* of another one if all its *elements* are subsets (or even equal) with respect to the corresponding ones in the larger *sequence* (or *elements* of larger index).

The majority of previously developed algorithms are based on the **Apriori** technique reviewed before, that as we have seen raises computational complexity issues when the database is huge and the patterns to be extracted are in great number and with a large variety of *items*. To recap, these problems are due to the step-wise candidate sequence generation, which involves the possible escalation to an enormous number of sequences (especially during the first steps of the algorithm, where we can have a lot of allowed combinations in the *join step*, since the *Apriori* principle is not easily applicable yet) and difficulties at extracting long sequential patterns (since before the algorithm has to extract before the numerous sequences of shorter length, and accordingly to scan the whole database at each iteration to calculate the *support* of the candidates).

Prefixspan aims to exploit the logic behind the *Apriori principle* and in the meantime alleviate the burden of the expensive candidate generation. It accomplishes so by focusing on finding frequent prefixes, since any complete subsequence can be built by developing a frequent prefix. This strategy reduces the computational complexity of the previously developed algorithms for *sequential pattern mining* since now the sequential patterns are developed by inspecting just the local frequent patterns (i.e. the postfixes with respect to the discovered frequent prefixes).
Now we define these concepts better.

A *sequence* is identified as a **prefix** of another one with a less or equal number

of *elements* if and only if:

1. all its *elements* are equal to the corresponding ones of the larger *sequence* (not considering the last one);
2. its last *element* is equal (or at least a subset) of the corresponding one of the larger *sequence*;
3. removing its last *element* from the corresponding one of the larger *sequence*, all the remaining *items* are alphabetically after.

A *subsequence* **x** is the **projection** of its *supersequence* **y** with respect to the *prefix* **z** if and only if:

1. the *projection* **x** has this *prefix* **z**;
2. there exists no *supersequence* **w** of the *projection* **x** that is a *subsequence* of the *supersequence* **y** and has also the same *prefix* **z**.

The *sequence* **j** is the **postfix** of the *sequence* **y** with respect to the *prefix* **z**, if it is composed by all the *elements* of the *projection* **x** after the corresponding ones of the *prefix* **z** (while we remove the last *element* of the *prefix* **z** from the corresponding one of the *projection* **x**).

To summarize, the logic behind the *Prefixspan* algorithm consists of:

1. Find first all the *sequential patterns* of length 1 (i.e. all the *items* of the *sequential dataset* that have a *support* greater or equal than the minimum *support* threshold, found with a single scan of the whole dataset);
2. Subdivide the entire list of *sequential patterns* according to the *prefixes* constituted by the *frequent items*;
3. Extract one by one these partitions of the *sequential patterns* by building the corresponding *projected datasets* with respect to these *prefixes* and mine each of them *recursively* (so again subdividing the sub-list of *sequential patterns* to be found according to the *prefixes* constituted by an increasing number of *items*).

So, for example, all the *sequences* in the *sequential* dataset which contain the *item* **x** are *projected* with respect to the *item* **x** in order to obtain the **x**-*projected dataset*, which consists of the *postfix* sequences with respect to the *prefix* **x**. Then, *recursively*, we scan the **x**-*projected dataset* for one time in order to find the totality of *sequential patterns* of length two and having as *prefix* the *item* **x** (the *support* of a certain *sequence* is the number of sequences contained in the *projected dataset*

for which it is a *subsequence*). Having done this, the algorithm continues again to build the *projected datasets* with respect to these new longer *prefixes* in order to extract longer *sequential patterns*.

This *problem partitioning* allows the *Prefixspan* algorithm to address the task *recursively*, so in a sort of *divide-and-conquer* approach, as opposed to the sequential one of the classical *Apriori* algorithm.
The **advantages** are that there is not anymore a stage of *candidate sequence generation*, since the longer *sequential patterns* are already extracted from the shorter ones and not from potentially useless patterns; moreover, the *projected datasets* keep reducing in size as the associated *prefixes* becomes longer, since there are less *postfixes* sequences (indeed, often just a little portion of *sequential patterns* grow very long in the dataset, and now we are not forced to run through a long sequence of highly computationally complex steps in order to find them).
As **drawback**, in the worst case, the algorithm has to build a *projected dataset* for each one of the extracted *sequential patterns*, and this can lead to computational bottlenecks.

2.2 Decision Tree

A **Decision Tree** is a *predictor*, inferred through a *supervised learning algorithm* (i.e. it is trained by means of humanly annotated datasets), which predicts the value of a label or a target variable assigned to an instance of the input space by walking through a path that goes from the *root node* to a *leaf* (or *terminal node*) of a tree.

Each *leaf* of the tree represents a **region** of the input space which has been identified by a set of *splitting rules*. Indeed, every *internal node* of the tree is associated with a splitting of the domain of one of the features that characterize the records to be classified. An example of this can be seen in the figure below, where in correspondence of each *internal node* a feature and a threshold has been chosen; then, the path will be in the direction of the right *branch* for the instances that have a value greater than the threshold for that feature, left direction otherwise. The built tree defines partitions of the input space, which are non-overlapping regions represented by high-dimensional rectangles. So, in the *test* phase, a new instance unseen by the algorithm during the training travels from the *root* to a *leaf*, ending its journey into one of these regions; then, the model assigns the *majority class* among the training records which belong to that area of the input space.

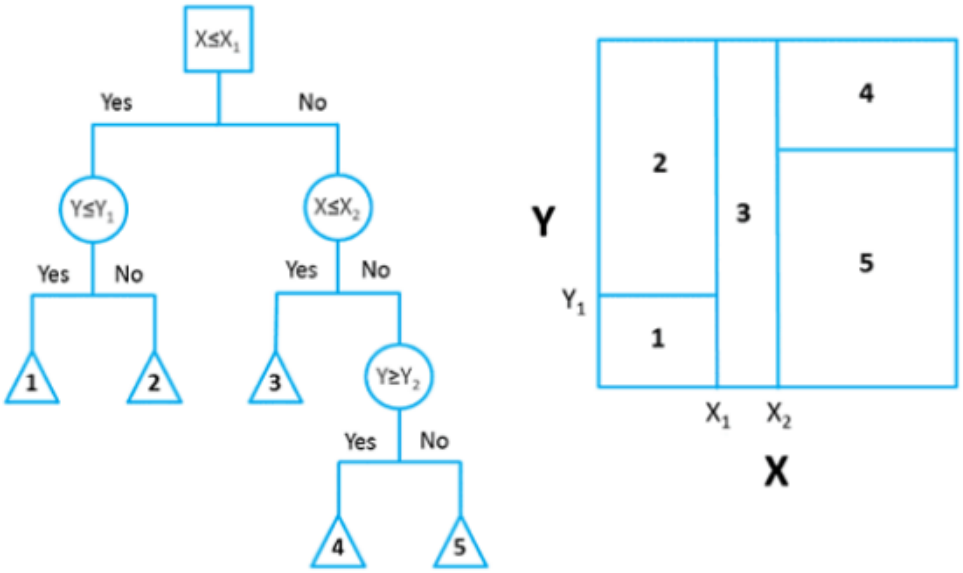

Figure 2.7: A decision tree and the associated splitting of the data space.

Intuitively, we can say that the objective of the algorithm is to build the best possible tree, meaning that it finds the optimal splitting of the input space in order to get regions where the training records are present predominantly in one of the classes. However, performing an *exhaustive search* of all the possible configurations is unfeasible from the computational point of view; moreover, a perfect tree for the training set would most likely incur in *overfitting* and so bad performances in the test set.
For this reason, a more practical *top-down greedy* approach is taken, called **recursive binary splitting**. Indeed, we proceed in the tree construction starting from the *root*, considering just two-branches splits. This process follows a *greedy heuristics*, meaning that at each step the algorithm considers the *best split* at that particular step of the tree construction: this could lead to sub-optimal decisions because we are not looking into the future to perform splits that could bring us to a better tree in a subsequent step, but in general it is a good approximation in order to build a tree in a computationally feasible way. There are various criteria which can be taken by the algorithm to determine the *best split*, and here we explore the most commonly adopted two.

Gini index is a measure of the *node impurity*, meaning that a high value stands for a node in which there is a huge promiscuity of class label: so the algorithm looks for the splitting that could give the biggest reduction of the index value. This index lies in the range $[0, 1 - \frac{1}{n_c}]$, where n_c = number of class labels, so in the case of binary classification a value of 0.5 represents a perfect equilibrium between the

classes into the node. The formula is reported below:

$$G = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) \tag{2.5}$$

where p_{mk} represents the proportion of training instances in the mth region which belongs to the kth class.
Another criterion is the **Cross-entropy**, given by:

$$D = -\sum_{k=1}^{K} p_{mk} \log p_{mk} \tag{2.6}$$

We *recursively* perform the algorithm of tree construction for each generated subtree until some *stopping criterion* is met (e.g. we end up with *leaves* which have less than a certain number of training records). However, growing up a tree of indefinite size could make us incur in *overfitting*: so, in order to get a tree which is more likely to generalize, we could try to stop the tree building in advance, setting a *maximum depth*.

To sum up, the **advantage** of trees is that it constitutes a very simple model, easily interpretable even by non-experts, explorable through its graphical display. Moreover, we do not have the need to create *dummy variables* in case of *categorical features*, since the algorithm is capable of managing the splits as they are. The main **drawback** is that generally they do not perform at the same level as other classification techniques and they are prone to *overfitting*.

As we have discussed, *Decision tree* is a pretty visualizable model, meaning that the structure and the characteristics of the tree diagram that we have been obtained by the algorithm in order to classify the test data can be easily shown. For example, for each node, it is possible to report:

- The *splitting rule* used to perform the node branching and lower the node promiscuity in terms of class labels;
- The value of the *node impurity* metric, that we want to be the lowest possible for the leaves of the *Decision tree* built by the algorithm, since they are used to determine the class label to be assigned to a sample for a prediction;
- The number of samples of the *train* dataset that have crossed a path inside the built *Decision tree* until a certain node;
- The class distribution, which is the number of samples belonging to the various class labels inside a certain node.

Different colors can be used to represent majority of samples belonging to a certain class label, while the tonality of these colors can stand for the value of *node impurity* among the various class labels present inside a certain node.

2.3 Random Forest

To cope with the problem of *overfitting* and improve the performances, **ensemble trees** methods have been explored, exploiting the approach of building multiple trees and combining their knowledge.

Random Forest makes use of the *bagging* technique, meaning that each one of the trees in our ensemble is trained on a different *bootstrapped* set of the training data (i.e. we perform a random sub-sampling of the training data with replacement, considering the records as uniformly distributed). Then, to get trees which are in some way *decorrelated*, the following technique is adopted: when building each decision tree, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors, considered again as uniformly distributed (a typical value for m is $\sqrt{p}$).
So, after having built the desired number of decision trees, a new test instance not seen by the algorithm during training travels inside each of the trees, getting a class prediction from each of them; then, the overall prediction will be the most predicted class among the ensemble (i.e. *majority voting*), as shown in the Figure 2.8.

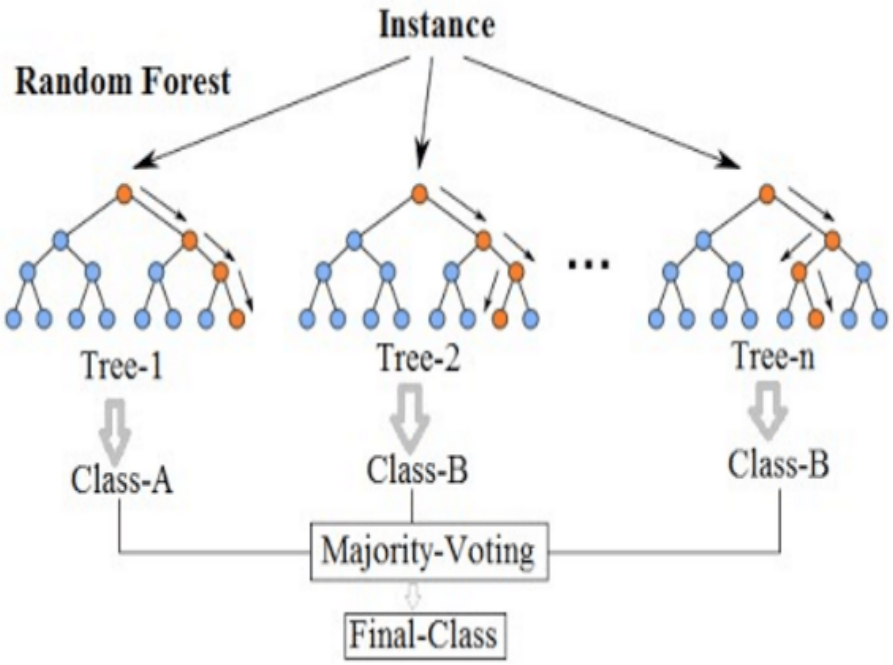

Figure 2.8: Visualization of Random Forest.

An **advantage** of this model is its robustness with respect to noise and *outliers*, since the prediction is a *majority voting* among *decorrelated* trees trained on *independent* sub-sets of the training data. Moreover, it can also be estimated a *global feature importance* to identify which are the most relevant for the classification task: for instance, it can be calculated the total amount of *node impurity reduction* given by a particular feature across all the trees. As **drawback**, better performances are obtained at the expense of the simplicity and *interpretability* of the model, since now we have a prediction given by an aggregation of hundreds of trees.

As we have discussed, *Random forest* can also provide insights in terms of interpretability, since it can be estimated a *feature importance* to identify which are the most relevant for the classification task.
For example, we can use as such metric of *feature importance* the *mean decrease in impurity* (MDI), which is the mean of accumulation of the impurity decrease within each tree belonging to the *ensemble*. A histogram could be used to easily detect:

- Which of the features dominate over the others in terms of MDI (resulting so the most discriminative features for the classification task at hand, as they are the most helpful ones in order to isolate across the *feature space* the samples belonging to the various class labels);
- Which of the features are irrelevant in order to distinguish the class labels

across the *feature space* (resulting so the less discriminative features for the classification task at hand, as they do not significantly contribute to the lowering of the *node impurity*);

- Which of the features share more or less a similar discriminative power (resulting so, in actionable words useful for the decision-maker, features that are equally decisive for the managerial planning).

Chapter 3

Methodology

3.1 Problem and solution description

As written in the title, in this thesis we deal with the assessment of the classification task to be performed over *spatio-temporal* data deploying an *associative classifier*.

In *machine learning*, the classification task consists of assigning a predefined set of labels to data samples contained into a dataset. In the case of our thesis, we have a specific typology of data which requires an appropriate handling to perform this task.
Spatio-temporal data are characterized by information distributed both in space and time, and so they are usually reported in the format of logfile dataset, which records with a certain frequency what happens in specific locations. As we can imagine, a huge variety of application fields generate such data: industrial production, where we can gather data about the status of the various machineries present in the factory during the entire production cycle; finance, where we can collect values of economic indicators over a time range from different geographical sources.
Nowadays, they are remarkably important in the sector of the *digital platforms*: this is a really huge source of this type of data, since the services that these digital applications are offering are distributed in time and space, such as the bike sharing service that we use as case study in this thesis. Another example of this can be a food delivery service, which keeps track of the historical data about the orders submitted by the users (with details about the exact times of the orderings and of the trips for the deliveries) and the associated spatial information such as the location of the users and restaurants and the position of the riders. All these information can be put together and exploited in order for the algorithm behind the digital application to match user's orders and riders in the most efficient way possible.

Usually, the classification task over this typology of data revolves around the detection of some kind of *event* which has happened somewhere within a specified time horizon. In this way, class labels are conditions determined by the status of the entities which represent the source of the data gathering over space and time. For example, we can train the classification algorithm in order to detect conditions of interest that can signal the breaking of a machine in a few time for purposes of *predictive manteinance*. The task can consists of distinguishing between multiple class labels, or instead detecting just the presence or the absence of a specific event (binary classification task with a *one vs all* approach).
Moreover, we have to adopt a data representation which is suitable for the classification algorithm that we want to deploy. In this thesis, we rely on an *associative classifier*, which requires the extraction of *association rules* from a dataset of transactional type, as detailed in Chapter 2. The strategy used in this work is to transform the original logfile into a sequence of events, so that it can be processed by a *sequential data mining* algorithm (such as **Prefixspan**) to get patterns that can be interpreted as the set of association rules at disposal of the classification algorithm. As detailed later in this Chapter, our aim is to get *spatio-temporally invariant* patterns in order to have a classification algorithm based on association rules which are robust and with more expressive power.
Anyway, we rely on this sequential representation (transposed into a tabular format) also for the other classification algorithms that we deploy (i.e. *decision tree* and *random forest*), in order to simplify the data representation by reducing the cardinality of the original data domains. Unlike the associative classifier, we do not have to explicitly extract patterns by means of a data mining algorithm, but instead the classification algorithm internally deduce a characterization of the class labels, such as the paths of a tree diagram.

Figure 3.1: Graphical representation of the proposed solution.

We graphically sum up the proposed solution in Figure 3.1:

1. **Data preprocessing**: as first phase, we have to get the sequential data representation by implementing an algorithm capable of transforming the original tabular dataset into a one composed of sequences of events which happen within a predefined spatio-temporal horizon;

2. **Pattern extraction from the training data**: having the sequential representation of the data at our disposal, we can deploy a sequential data mining algorithm to extract patterns from the training partition of our dataset, necessary to implement the associative classifier;

3. **Classification of the test data**: as last phase, we deploy our algorithm to perform an associative classification (i.e. interpreting the patterns extracted by the sequential data mining algorithm as association rules to be matched with the samples to be classified) over the test partition of our dataset, which has gone through the same preprocessing step of the training one;

In the rest of this Chapter, we see the details of each one of the solution stages.

3.2 Dataset

The data source for this thesis is [5], a transformed version of the open data shared by **Bay Area Bike Share**, available here [6]. For our purposes, we make use of these three datasets:

- **Station** dataset, which contains information about the bike stations;
- **Status** dataset, samples about the availability of bikes and docks for a given station and time;
- **Trip** dataset, which keeps track of trips made by individual bikes.

3.2.1 Station dataset

This dataset consists of 70 entries and 7 features:

1. **id**, a numerical id which identifies the station;
2. **name**, with which the station is known;
3. **lat**, the latitude of the station's location;
4. **long**, the longitude of the station's location;
5. **dock_count**, the number of dock at disposal of the station;
6. **city**, where the station is placed;
7. **installation_date**, the date since the station is active.

3.2.2 Status dataset

This dataset consists of 71.984.434 entries and 4 features:

1. **station_id**, the numerical id identifying the station;
2. **bikes_available**;
3. **docks_available**;
4. **time**, the date and time for which the status of the station is recorded, in the format yyyy/mm/dd hh:mm:ss;

3.2.3 Trip dataset

This dataset consists of 669.959 entries and 11 features:

1. **id**, a numerical id which identifies the trip;
2. **duration**, in seconds;
3. **start_date**, when the trip begins;
4. **start_station_name**, where the trip begins;
5. **start_station_id**;
6. **end_date**, when the trip ends;
7. **end_station_name**, where the trip ends;
8. **end_station_id**;
9. **bike_id**, a numerical id which identifies the individual bike;
10. **subscription_type**, typology of the user whom rides the bike.

3.2.4 Training, test and validation split

A proper training procedure for *machine learning* algorithms requires a splitting of the dataset into different partitions, in order to assess fairly the performances of the models. Indeed, we subdivide the original datasets into training and test ones, to calculate the classification evaluation metrics predicting over samples that have not been processed by the algorithm during its training stage. This simulates real-world applications, where we do not have all the data at our disposal and we are required to predict new ones.
In order to tune the *hyperparameters* (i.e. parameters of the algorithms that

need to be set in advance with respect to the training stage) of the classification models, we further split the training set to extract a validation partition. Doing so, we avoid incurring in **overfitting** over the test set, ending up selecting a set of *hyperparameters* which is optimized on the specific test partition that we have selected, and so not able to generalize well.
Data from *status* and *trip* go from 29/08/2013 to 01/09/2015. We choose to keep almost the $\frac{2}{3}$ of data for the training partition: dealing with *time series*, we have to make sure that the samples are not selected randomly, but instead follow a chronological order (past data into the training partition and future data into the test one). In this way, data until 31/12/2014 are contained in the training partition, while the rest in the test one; in the same way, data from 01/08/2014 are part of the validation partition.

For some experiments, we do not keep just the whole selection of data, but instead we subdivide the dataset partitions into timeslots of 4 hours (00-03, 04-07, 08-11, 12-15, 16-19, 20-23), in order to better assess the models performances in the various parts of the day and their different trends. Indeed, each timeslot is characterized by its own type of traffic, determined by the necessity of people to go to work, university, gym... leaving some timeslots inevitably with more stability in the stations' status, while the others have a more frenetic pace of changes.

3.3 Algorithm to generate sequences of events

In order to deploy an *associative classifier*, we have to transform the original *status* dataset into a structure which is more suitable to a *transactional* format, having so the possibility to extract the *frequent itemsets* and generate the *association rules*.

As data representation, we set our *items* to be of the type "Event_Station_Time" in order to rely on *association rules* constituted by *spatio-temporally invariant* patterns (whose points of strength are pointed out later with the aid of graphical examples). As said in the Introduction, we want to keep *relative* references to stations' locations and time. So, instead of identifying a station by means of its *id*, we make use of the *discretized distance* between it and a *reference station*, considering in which one of the *concentric circles* around the *reference station* the station falls. As for the time, we do not use the actual date and hour, but instead we discretize it by means of *temporal windows*, so that each record of the dataset falling into a window happens in the same *discretized instant*.
The last step for our alternative data representation consists in defining some *events* related to the status of a station. We have chosen to adopt these ones:

- **Full**, which represents the situation when the number of docks available is

equal to 0;

- **Almost full**, which represents the situation when the number of docks available is less or equal than a certain threshold;
- **Empty**, which represents the situation when the number of bikes available is equal to 0;
- **Increase**, which represents the situation when the number of bikes available is greater than the one of the previous timestamp;
- **Decrease**, which represents the situation when the number of bikes available is less than the one of the previous timestamp.

Moreover, we have decided also to take note of the **Normal** state, which represents the situation when the critical condition has not happened within a certain *time window* for a specific station. This is especially necessary in order to not lose the type of *sequences* for which we have a static situation (i.e. none of the *events* of interest has happened inside the *time window*, which means that the related records of the *status* dataset would not contribute to the *sequence* generation, resulting in an incomplete representation of the data at our disposal), and so have an accurate representation of the negative class. So, for these classification purposes, we keep track of this type of *event* just for the *reference station* during the "instant 0".

So *status* dataset records are subdivided into *time windows* of a certain length, and all the stations that are inside a certain window will assume the role of *reference station*, in order to represent in a *relative* way the location of all the other stations. We also have to define the time horizon for the patterns we are going to extract (i.e. the number of *time windows* that are inspected for each *transaction*), since in turn one of the *time windows* will become the reference one (so the "*discretized instant* 0") and each *sequence* (i.e. the *transaction*) will keep track of all the *events* of interest that happen in the neighborhood of the *reference station* within the time horizon that we have set.

Before we run the algorithm, we have to define a set of parameters, representing the kind of *sequences* that will be generated:

- **Temporal_threshold**, which is the length of the *time windows*, expressed in minutes;
- **Spatial_threshold**, which is the radius of the *concentric circles* drawn around the *reference station*, expressed in kilometers;
- **Temporal_steps**, which is the number of *time windows* taken into account for each generated *sequence*;

- **Spatial_steps**, which is the number of *concentric circles* that we consider when we define the neighborhood of the *reference station* by the **Neighbor_type** *radius*;

- **Support**, which is the minimum threshold used for the extraction of the *association rules*;

- **Max_patt_len**, which is the maximum length of a sequence (i.e. the total number of *items*);

- **Cities**, that is the set of cities on which the generation of *sequences* and the extraction of *rules* is performed;

- **Event_types**, which is the set of *events* that are taken into account during the generation of the *sequences*;

- **Neighbor_type**, which defines how the algorithm selects the neighbors of the *reference station*;

- **Top_n**, which is the number of selected neighbors of the *reference station* when the **Neighbor_type** is set to *incoming*;

- **Ignore_set**, which is the set of *events* that will be excluded when projecting the *sequences* with respect to the *reference station*;

- **Reproj_mode**, which defines if, given a *time window* as reference, the time horizon of the algorithm will be backward or forward;

- **Absolute_mode**, which defines if representing the stations by means of their *Id*'s or the *discretized distance* with respect to the *reference station*.

So, in addition to the setting of a time horizon for the *sequences* generated by the algorithm, we define a neighborhood of the *reference station* for which the *events* will be recorded, that could depend from the proximity with respect to the *reference station* (so a spatial horizon) or could be constituted of stations that have a certain "link" with the *reference station*. The **Neighbor_type** can be of this kind:

1. **Radius**, meaning that the neighborhood of the *reference station* will consist of all the stations located within the **Spatial_steps** that have been set.
   For example, if this parameter is set to 5, a *sequence* will report all the *events* belonging to the **Event_types** set that have happened in all the stations placed within a circle of radius **Spatial_threshold** $*$ **Spatial_steps**.

2. **Incoming**, meaning that the neighborhood of the *reference station* will consist of the **Top_n** stations which have the highest number of records into the *trip* dataset (i.e. records where the **Start_station_id** is the neighbor, while the **End_station_id** is the *reference station*). Doing so, we are selecting the stations from which the highest number of bikes is arriving to the *reference station*, regardless of their distance with respect to the *reference station*.

When adopting the **Neighbor_type** parameter to *incoming*, we also omit to report in the *sequences* the **decrease** status for the *reference station* and the **increase** status for the neighbors when the critical condition that we want to classify is the **full** one, since to predict this status is much more useful to know when the number of bikes in the *reference station* is increasing and when the number of bikes in the neighbors is decreasing.
For what concerns the **Reproj_mode**, we set this parameter to "past" in order to use the information about the *events* that happened in the stations belonging to the adopted **Neighbor_type** during the preceding **Temporal_steps** *time windows* with respect to the reference one. We also set the **Absolute_mode** parameter to "false" in order to use instead a representation for the neighbor stations that is relative to the *reference station* (i.e. the *discretized distance* from the neighbor), extracting in this way patterns which are **spatio-temporally invariant**.
Further details about our algorithm deployed to transform the original logfile dataset into a sequential one (appropriate to be used as input for the *sequential data mining* algorithm) are reported in [7].

3.3.1 The spatio-temporal invariance

We underline the expressive power of *spatio-temporally invariant* patterns by showing a graphical representation, as reported in Figure 3.2.

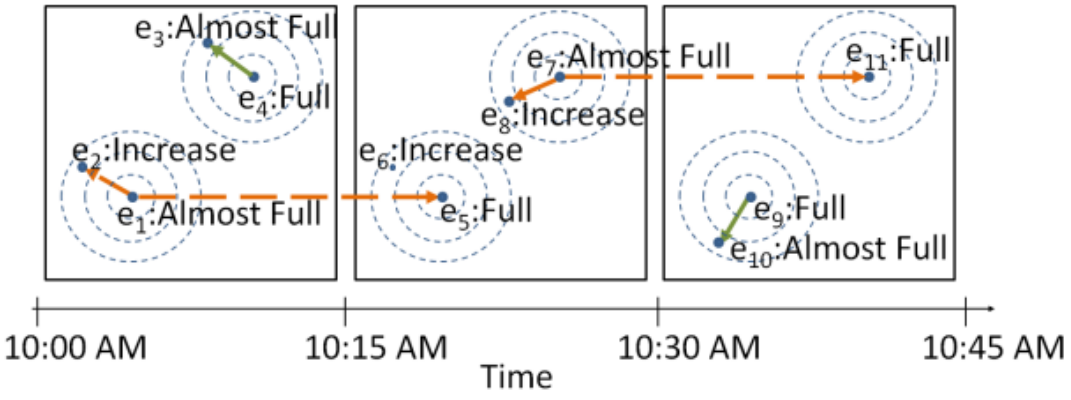

Figure 3.2: Graphical representation of *spatio-temporally invariant* patterns.

As we can clearly see from the image above, in this case the parameters of our algorithm which generates the *sequences* are set as follows:

- **Temporal_threshold**: 15 minutes, as we can see that each *time window* has this duration;
- **Spatial_threshold**: 0.1 kilometers, which is the length of the radius of the *concentric circles* centered in the *reference station*;
- **Temporal_steps**: 3, as we can see that the *time horizon* under analysis is composed by three *time windows* of **Temporal_threshold** duration (where the first one is the *reference window* from when the *time horizon starts*);
- **Spatial_steps**: 3, as we can see that the *spatial horizon* of the algorithm spans over three *concentric circles* centered in the *reference station*;
- **Event_types**: Full, almost full and increase are the kind of *events* that we want the algorithm to detect;
- **Neighbor_type**: Radius, as we can see that the algorithm looks at the nearest *stations* with respect to the *reference station* (i.e. the ones falling within the *spatial horizon* set by **Spatial_steps**);
- **Ignore_set**: Increase, as the building of the *sequences* revolves around the occurrence of *full* and *almost full* in the *reference station*;
- **Reproj_mode**: Future, as we can see that the *time horizon* moves in forward direction with respect to the *reference window* (from 10:00 AM to 10:45 AM);
- **Absolute_mode**: False, as we want the algorithm to highlight *spatio-temporally invariant* sequential patterns.

Putting *sequences* generated by our algorithm into graphical form makes more evident what do we mean by *spatio-temporal invariance* and its implications.
Stations are represented as points inside the *concentric circle* centered in the *reference station* where they fall into; the *reference station* is the point placed at the center of the smallest *concentric circle* and it is indicated as the "station zero", since it is the only *station* located into the hypothetical *concentric circle* of radius zero.
In correspondence of the points, we have the *items* e_i belonging the *sequences*, which are the triplets "Event_Station_Time". So we have the indication of the *event*, the rectangular box that indicates in which *time window* the *event* has occurred, and the spatial location of the *station* indicated by the *concentric circle*

centered in the *reference station* in which they fall into.
The *spatio-temporally invariant* patterns that we want to extract (and then to use in order to perform *classification*) are specific co-occurrences of *items* distributed in time and space that happen "frequently" (according to the minimum *support* threshold) inside the generated *sequences* of triplets. The continuous lines symbolize a co-occurrence of *items* between the *reference station* and another one, whereas the dashed lines symbolize a co-occurrence of *items* associated with the *reference station* in different time slots; the different colors of the arrows stand for distinct *spatio-temporally invariant* patterns that are going to be extracted by the algorithm:

- **Orange** pattern: this *spatio-temporally invariant* pattern is characterized by a *reference station* which is in an *almost full* condition and another *station* in the area of the third *concentric circle* which is in an *increase* condition during the *reference time window*, and then in the next *time window* the *reference station* turns into a *full* condition (this co-occurrence of *items* happens two times: one including the *items* e_1, e_2 and e_5, and the other one including the *items* e_7, e_8 and e_11).

  This type of *spatio-temporally invariant* pattern that has been extracted by the algorithm can be translated into words useful for the fleet manager of the bike-sharing service in order to plan supply routines for the *stations*. The information contained in this pattern is that if a certain *station* is in an *almost full* condition (i.e. the situation when the number of docks available is less or equal than a certain threshold) and at least one of its neighbor *stations* in the area between 200 and 300 meters is in an *increase* condition (i.e. the situation when the number of bikes available is greater than the one registered during the previous detection) within fifteen minutes, then that same *station* will turn into a *full* condition (i.e. the situation when the number of docks available is equal to zero).
  These correlations of *events* distributed in time and space can be useful to identify the situations when a certain *station* is about to saturate their docks spots (based on its current status and the one of at least one of its neighbors located in a precise range of distance from itself): so the fleet manager of the bike-sharing service can intervene by redistributing a certain number of bikes from this station to another one which lacks them.

- **Green** pattern: this *spatio-temporally invariant* pattern is characterized by a *reference station* which is in a *full* condition and another *station* in the area of the third *concentric circle* which is in an *almost full* condition during the *reference time window* (this co-occurrence of *items* happens two times: one including the *items* e_3 and e_4, and the other one including the *items* e_9 and e_{10}).

This type of *spatio-temporally invariant* pattern that has been extracted by the algorithm can be translated into words useful for the fleet manager of the bike-sharing service in order to plan supply routines for the *stations*. The information contained in this pattern is that a certain *station* is in a *full* condition (i.e. the situation when the number of docks available is equal to zero) when at least one of its neighbor *stations* in the area between 200 and 300 meters is in an *almost full* condition (i.e. the situation when the number of docks available is less or equal than a certain threshold) within fifteen minutes. These correlations of *events* distributed in time and space can be useful to identify the situations when a certain *station* is about to saturate their docks spots (based on the current status of at least one of its neighbors located in a precise range of distance from itself): so the fleet manager of the bike-sharing service can intervene by redistributing a certain number of bikes from this station to another one which lacks them.

We can see how the purpose of the algorithm is to detect, between all the tracked status of the various *stations* over the days, which *sequences* of *events* are *recurrent* (i.e. repeated many times inside the records of the *status* dataset).
The detection of these *sequential patterns* is possibile just because we are representing all the spatial and temporal informations about an *event* in a *relative* way: the algorithm indicates the *stations* by means of a *discretized distance* with respect to a *reference station* (counting in which one of the *concentric circles* centered in the *reference station* they fall within), and indicates the time by means of a *discretized timeline* for which each timeslot of duration **Temporal_threshold** is seen as a unique instant of time (i.e. all the *events* happened within the same *time window* are seen as if they have happened at the same time). This representation method is fundamental in order to see the *events* during the day "as they are", without the specificities given by an *absolute* method of representation (i.e. the algorithm indicates the *stations* by means of their unique numerical*id*, and indicates the time by the exact date and hour):

- The peculiar identity of each *station* is replaced by taking the point of view of any *station* in which has occurred a *trigger event* (i.e. the ones belonging to the **Event_types** set, but not to the **Ignore_set** one) and then projecting this vision to all the nearby *events* contained in the delimited (by the parameters of the algorithm) *spatio-temporal* landscape under analysis.

- The precision of the time reference is replaced again by taking the point of view of that *trigger event* and placing the "instant zero" in the *time window* in which it has happened, so that we can track the time in terms of *discretized timesteps* proceding from the *reference time window*.

Doing so, we can actually highlight *spatio-temporally invariant* patterns, because

the algorithm can detect a certain "repetitive structure" of *events successions* (such as the different colors in Figure 3.1). Instead, in an *absolute* framework, this would be more difficult since there is much more variability of values assumed by *stations* and time. The *invariance* is then given by the fact that the pattern can be placed in different space and time but still being the same identical pattern, as in a *relative* representation method what it counts it is just the configuration of the spatial and temporal gaps between the *events* in a pattern, and not their exact location and timing.

Chapter 4

Experimental section

4.1 Setting and Metrics

Before showing the models deployed for the classification tasks and the related results, we first specify the framework adopted to perform the training.

4.1.1 Classification task

The dataset analyzed in this thesis is about a bike-sharing service, which requires users to take the vehicle from certain locations all over the city called *stations* and then, at the end of their trip, park it in another *station* (or also the same where they had pick up the bike at the beginning). This type of services needs that the owners of the digital platform are aware of when and where bikes could be useful, and from this fundamental aspect depends the success of the application: indeed for example, they have to avoid situations where a lot of users find out that all the *stations* nearby them are unable to satisfy their demand, whereas other far *stations* are full of bikes but without incoming requests.

To do so, there should be a planned routine of bike supply during the different parts of the day, to understand where to pick up the vehicles and where to put them and with which timing.
For this purpose, we decided to address the task of *binary classification* between a critical condition associated to the *reference station* (such as the totality of docks inside the *station* full of bikes, or instead the docks without the presence of vehicles) and the absence of it (with the first case denoted as the *positive* class and the other as the *negative* one). The data representation differs depending on the model that we are using to classify.

Associative classifier

To appropriately take advantage of this classifier, we need to generate *sequences* (as described in the Chapter 3) from the *status* dataset: the ones created from the *test* partition will be used to be classified and to evaluate the performances of the algorithm, while the ones from the *train* partition are the input for the **Prefixspan** algorithm to execute the *sequence mining* and extract the patterns useful to assign the class during the prediction stage. The selection of the rules adopted during the classification works as follows:

- Application of the **Prefixspan** algorithm to perform the *sequence mining* task over the *status* dataset. The extracted *spatio-temporally invariant* patterns are then interpreted as our set of *association rules* by considering the *body* rule as the *time windows* from the past, while the *head* rule is the *time window* for which we want to perform the prediction of a critical condition for the *reference station* (as explained in Chapter 2);
- Selection of the rules which contain at least one *event* associated to the *reference station* and happened at the "instant zero";
- Filtering out the rule list according to the critical condition associated to the *reference station* and happened at the "instant zero" that we are interested in classifying (keep just the rules which contain it, and not having *events* of interest happened in different *stations*);
- Setting of a minimum *confidence* threshold to select the rules that have a certain "strength";
- Setting of a minimum *support* threshold in order to keep just a certain number of rules (setting so an order of magnitude for the adopted rules).

So, we have to use an algorithm which is capable of extracting *association rules* from such a database. Actually, to achieve this goal, we rely on an algorithm which performs *sequence mining* to extract the spatio-temporal patterns of our interest, and then we interpret them as *association rules*.
Since we want to predict a situation of criticality for a station based on the conditions of its neighborhood during a predefined time horizon in the past, we identify as *body* of the rule the set of events temporally localized in the past *time windows*, while the *head* of the rule is identified as the set of events temporally localized in the last *time window* (i.e. the *time window* for which we want to predict a critical condition for a certain station).

Having done so, we classify one by one the sequences generated from the *test* partition of the *status* dataset by looping through the obtained list of rules to search for a certain number of rules (i.e. the minimum *matching* threshold that

we have set) which *cover* the analyzed instance. In other words, a *sequence* is classified as belonging to the *positive* class if it contains a number of selected rules greater or equal than the minimum *matching* threshold (so that they are *subsets* of it, except for the "instant zero", which is the timeslot to be predicted and so we do not look for pattern matching in that period). Instead, a *sequence* actually belongs to the *positive* class if it contains the critical condition of interest associated to the *reference station* at the "instant zero".

Other classifiers

To deploy classifiers such as **Decision tree** and **Random forest**, we need tabular data with couples attribute-value. Our choice is to get them from the already generated *sequences* from the *status* dataset (the ones used to extract the *association rules*). So, the data representation will be different depending if the neighborhood of the *reference station* is built in the *radius* or the *incoming* mode:

- **Radius**: the input data for the *classification* algorithms have a column for each combination of *event*, *station* and *time window* as set by the parameters **Event_types**, **Spatial_steps** and **Temporal_steps** of the *sequence generation* algorithm (except for the "instant zero", for which we just report the class label, since we do not want to use features belonging to the same *time window* of the class label for making predictions);

- **Incoming**: the input data for the *classification* algorithms have a column for each combination of *event* and *time window* related to the *reference station* as set by the parameters **Event_types** and **Temporal_steps** of the *sequence generation* algorithm (except for the *decrease event* in case the *positive* class is associated with the *full* condition), and a column for each *time window* related to the whole neighborhood (reporting if there is at least one *station* belonging to the neighborhood of the *reference station* which has been in the *decrease* condition during that specific *time window*). Again, for the "instant zero", we just report the class label, since we do not want to use features belonging to the same *time window* of the class label for making predictions;

All these columns are of *binary* type, signaling if the situation represented by it has occurred. Having done so, we classify one by one the sequences generated from the *test* partition of the *status* dataset by applying the predictions performed by the models trained on the *training* partition of the *status* dataset. Again, an instance actually belongs to the *positive* class if it presents the critical condition of interest associated to the *reference station* at the "instant zero".

4.1.2 Evaluation metrics

Here we revise the most commonly adopted metrics for the evaluation of a classification task. As notation, we define:

- **TP** (True positives): the number of records which have been predicted to belong to the positive class and that actually belong to that class;
- **TN** (True negatives): the number of records which have been predicted to belong to the negative class and that actually belong to that class;
- **FP** (False positives): the number of records which have been predicted to belong to the positive class, but actually belong to the other class;
- **FN** (False negatives): the number of records which have been predicted to belong to the negative class, but actually belong to the other class;

In our case, we consider the *critical event* that happened in the *reference station* as the positive class, while the absence of this type of *event* as the negative one.
Accuracy represents the proportion of correct predictions over all the ones that the model has made, regardless of the class, and is given by:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Actually, in case of unbalanced datasets, we can get fooled by this metric because we can get good scores simply performing very well on the *majority class*, since correct predictions with respect to that class will have a higher weight in the formula: if we have a strongly unbalanced dataset, we could even predict all samples to belong to the *majority class* and still obtaining a value close to 1.
Precision represents the proportion of correct predictions with respect to a specific class, and is given by (considering the positive class):

$$P(+) = \frac{TP}{TP + FP} \tag{4.2}$$

In our particular case, we can say that in principle we are more interested in obtaining a high value for the positive class, since we do not want to abuse interventions in stations that do not require them, wasting resources that would have been much more useful in other locations.
Recall represents the proportion of correctly predicted samples over all the ones actually belonging to that class, and is given by (considering the positive class):

$$R(+) = \frac{TP}{TP + FN} \tag{4.3}$$

In our particular case, we can say that in principle we are more interested in obtaining a high value for the positive class, since we want to correctly detect the *sequences* in which a critical *event* has happened in the *reference station* at the "instant zero", limiting as much as possible to miss an intervention in a station in critical status that needs it.
F1 score represents the harmonic mean between **Precision** and **Recall** for a more balanced summarization of model performance., and is given by:

$$F1(+) = 2\frac{P(+)R(+)}{P(+) + R(+)} \tag{4.4}$$

The class-wise metrics can be aggregated in order to inspect the overall performance of the classifier with respect to both the classes: we have the *macro average*, which is simply the arithmetic mean of the metric values for the two classes, and the *weighted average*, where each metric value is weighted by the proportion of that class in the dataset. In our case, since we are dealing with an unbalanced dataset, and we are more interested in the performance with respect to the *minority class*, we prefer to consider the *macro average* since it does not give a higher weight to the *majority class* and considers equally each class, regardless of the number of samples that belong to it. In particular, for the *hyperparameters tuning*, we select for each model the configuration which obtains the highest *macro averaged F1 score* and the configuration which obtains the highest *macro averaged Precision score* over the *validation* partition.

An useful way to get an overview of the classifier performance is the so called **confusion matrix** which allows to instantly spot the amount of TP, TF, FP, FN and easily calculate the classification metrics. Following the convention for axes adopted by the *sklearn* library, each entry $C_{i,j}$ of the matrix represents the number of records which have been predicted to be part of class j and actually belong to class i: so we have $C_{0,0}$ = TN, $C_{1,0}$ = FN, $C_{1,1}$ = TP and $C_{0,1}$ = FP.

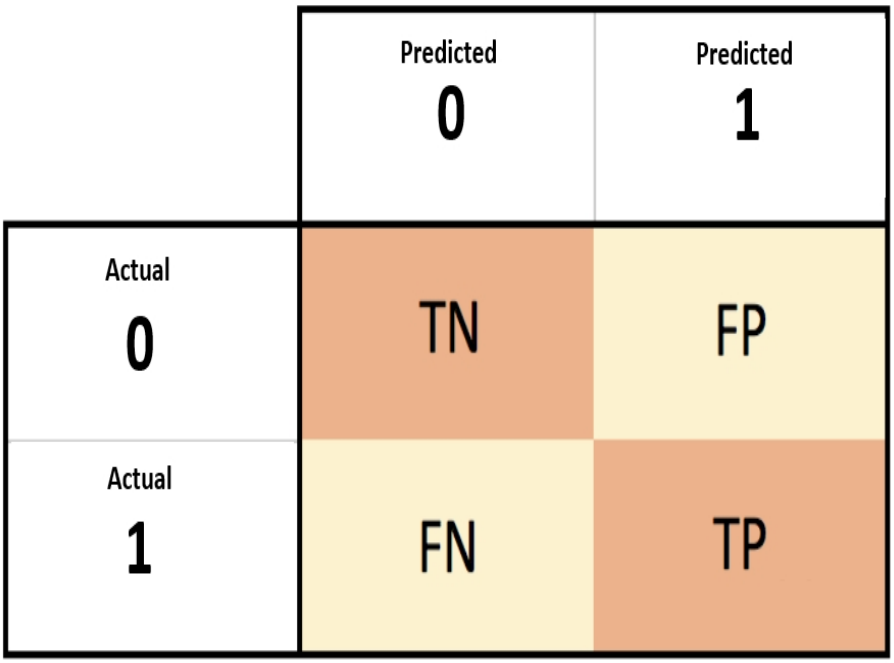

Figure 4.1: Example of a *confusion matrix* in case of *binary classification*.

4.1.3 Hyperparameters tuning

As it is known, classification models have a lot of **hyperparameters** (i.e. model parameters which have to be set in advance and are not inferred by the training procedure). The *sklearn* library sets a default value for each of them, but instead of this naive approach, we can try to tune the value of some of them to detect the model configuration that is more likely to perform better in predicting the test set. The problem of *hyperparameters tuning* is that, if we directly assess the performance of the various configurations with respect to the test set, we could incur into *overfitting* towards the particular set that we are using for the final evaluation, failing our objective to find a model capable of generalizing and obtaining satisfactory results for each data instance of the task at hand.

To deal with this issue, we adopt a training procedure which consists in splitting the training set to get a *validation* partition, and select the configuration of *hyperparameters* that performs the best on it (with respect to the evaluation metrics specified in the previous paragraph). Once we have selected it, we train the algorithm on the totality of the *training* data (so on the *training* partition of the *status* dataset as it was before performing the splitting into *train* and *validation* partitions).

Decision tree

The *hyperparameters grid search* for the **Decision tree** algorithm involves parameters which have an impact towards the construction process of the tree structure used by this algorithm to classify the samples (i.e. determine whether in a certain moment of the day, a certain station founds itself into a critical condition):

- The **min_samples_split** parameter, which is the minimum number of samples required to *split* an internal node. Otherwise, the *splitting* process involving the search of the optimal feature and optimal feature domain subdivision for that particular algorithm step (in order to optimally partition the *feature space* in which the data samples lie and so better isolate the two typologies of class label) is stopped and the node simply becomes a *terminal node* (i.e. a *leaf*).

- The **criterion** parameter, which is the index metric by which the building tree algorithm decides what is the *best split* for that particular step of the process.

The grid is composed as follows:

- *min_samples_split*: [0.02, 0.05, 0.1, 2, 10];

- *criterion*: [gini, entropy].

In addition to this *hyperparameters grid search* for the **Decision tree** algorithm, we also fix the *random_state* parameter, which controls the randomness associated with the construction of the tree diagram. Indeed, the *splitter* parameter (which is the strategy used by the algorithm to choose the *split* at each node of the tree diagram), can be set in two ways:

- **Best**, meaning that the *splitting* process involves the search of the optimal feature and optimal feature domain subdivision (among the selected *max_features*) for that particular algorithm step, in order to optimally partition the *feature space* in which the data samples lie and so better isolate the two typologies of class label;

- **Random**, meaning that during the *splitting* process the algorithm loops through the selected *max_features* and chooses a random *threshold split* for the feature domain subdivision, in order to optimally partition the *feature space* in which the data samples lie and so better isolate the two typologies of class label.

Although we choose to set the *splitter* parameter to "best", the *max_features* are always randomly permuted at each *split*, so introducing even in this case randomness into the algorithm. As a consequence of this, in order to obtain a deterministic behaviour during the fitting of the algorithm, *random_state* has to be fixed to an integer number.

Having selected the best configuration, we show the achieved performances by reporting the *confusion matrix* and the *classification metrics*.

Random forest

The *hyperparameters grid search* for the **Random forest** algorithm involves parameters which have an impact on the construction process of each one of the tree structures belonging to the *ensemble* used by this algorithm to classify the samples (i.e. determine whether in a certain moment of the day, a certain station founds itself into a critical condition), by taking the *majority vote* with respect to the predictions made by each one of the tree structures belonging to the *ensemble*:

- The *n_estimators* parameter, which is the number of *Decision trees* that the algorithm is going to build in order to then take the *majority voting* for the predictions made by the *ensemble*, where a high number should improve the model *robustness*;
- The **min_samples_split** parameter, which is the minimum number of samples (belonging to each one of the different *bootstrapped* datasets extracted from the *train* partition of the original *status* dataset used in the training process of each one of the trees in our *ensemble*; i.e. we perform a random sub-sampling of the training data with replacement, considering the records as uniformly distributed) required to *split* an internal node (for each one of the tree structures belonging to the *ensemble*). Otherwise, the *splitting* process involving the search of the optimal feature (among the random selection of m predictors that has been selected by the algorithm as *split* candidates from the full set of p predictors, considered again as uniformly distributed, during the building process of each one of the *Decision trees* belonging to the *ensemble*, each time a split in a tree has to be considered) and optimal feature domain subdivision for that particular algorithm step of the tree construction (in order to optimally partition the *feature space* in which the data samples lie and so better isolate the two typologies of class label) is stopped and the node of that particular tree inside the *ensemble* simply becomes a *terminal node* (i.e. a *leaf*).
- The **criterion** parameter, which is the index metric by which the algorithm that builds each one of the *Decision trees* belonging to the *ensemble* decides

what is the *best split* for that particular step of the process.

The grid is composed as follows:

- *n_estimators*: [10, 50, 100, 150, 200];
- *min_samples_split*: [0.02, 0.05, 0.1, 2, 10];
- *criterion*: [gini, entropy].

In addition to this *hyperparameters grid search* for the **Random forest** algorithm, we also fix the *random_state* parameter, which controls the randomness associated with the construction of the *ensemble* of tree diagrams. Other than the randomness brought by the *splitter* parameter (which is the strategy used by the algorithm to choose the *split* at each node of each one of the tree diagrams belonging to the *ensemble*, and incorporates randomness in both their possible settings), there are other elements in this algorithm that put randomness in its execution (i.e. the *bootstrapping* sampling procedure and the feature sampling described in Chapter 2). As a consequence of this, in order to obtain a deterministic behaviour during the fitting of the algorithm, *random_state* has to be fixed to an integer number.

Having selected the best configuration, we show the achieved performances by reporting the *confusion matrix* and the *classification metrics*.

4.1.4 Layout of the results

Here we explain how we want to report the carried experiments and the structure and conventions for the tables of numerical results.

We use a paragraph for each typology of *positive* class label that we have used for the *classification* task conducted by means of the various *machine learning* algorithms (i.e. *Associative classifier*, *Decision tree* and *Random forest*), and in a subparagraph we show the tables of numerical results for the specific value of the **Neighbor_types** parameter, which determines the extraction of the *spatio-temporally invariant* patterns and the data representation of the records to be classified.

For the different *machine learning* algorithms, the tables of numerical results report the following configurations:

- **Associative classifier**: we report the minimum *confidence* threshold associated with the *spatio-temporally invariant* patterns deployed for the *classification* task, and the minimum *support* threshold to select only a certain number of them. We also report the minimum *matching* threshold (i.e. the minimum number of "valid" *association rules* to be found in order to predict a record to belong to the *positive* class label);
- **Decision tree, Random forest**: we report the results obtained by these algorithms with the default configuration of *hyperparameters* as set by the *sklearn* library, and the results obtained by these algorithms with the optimal configuration of *hyperparameters* as selected after the *validation* procedure described in the subparagraph 4.1.3 over the evaluation metrics *precision* and *f1 score* (considering their *macro average*).

So, in the table of numerical results, the convention for the columns is:

- **A**: the value of the *accuracy* metric achieved by each algorithm;
- **P+**: the value of the *precision* metric with respect to the positive class achieved by each algorithm;
- **P-**: the value of the *precision* metric with respect to the negative class achieved by each algorithm;
- **R+**: the value of the *recall* metric with respect to the positive class achieved by each algorithm;
- **R-**: the value of the *recall* metric with respect to the negative class achieved by each algorithm;
- **F+**: the value of the *F1 score* metric with respect to the positive class achieved by each algorithm;
- **F-**: the value of the *F1 score* metric with respect to the negative class achieved by each algorithm;
- **TP**: the quantity of *true positives* detected by each algorithm;
- **TN**: the quantity of *true negatives* detected by each algorithm;
- **FP**: the quantity of *false positives* detected by each algorithm;
- **FN**: the quantity of *false negatives* detected by each algorithm;
- **ST**: the minimum *support* threshold to select only a certain number of *association rules*;

- **CT**: the minimum *confidence* threshold associated with the *spatio-temporally invariant* patterns deployed for the *classification* task;
- **#R**: the quantity of *association rules* through which the *classification* algorithm loops in order to find the necessary *matchings* to decide whether to classify a record as belonging to the *positive* class label;
- **TT**: the value set for the **TEMPORAL_THRESHOLD** parameter;
- **TZ**: the *time zone* of the *status* and *trip* datasets analyzed by the algorithm (i.e. a range of hours or the entire day time);
- **Alg**: the algorithm used to classify the records;
- **Val**: the evaluation metric considered during the *validation* procedure (i.e. *macro average* of *precision* or *f1 score*);
- **MT**: the minimum number of "valid" *association rules* to be found in order to predict a record to belong to the *positive* class label.

In addition to this, we report the most significant *association rules* (in terms of *confidence* and *support*) extracted by the algorithm of *spatio-temporally invariant patterns mining*, separately for each *time zone* (i.e. a range of hours or the entire day time).

4.2 Positive class: Full condition

In this typology of experiment, we identify as *positive* class the case in which the *reference station* has been in a **Full** condition during the "instant zero" (i.e. the *time window* considered as *reference* during the generation of the *sequences*), whereas as *negative* class the case in which this has not occurred.
The parameters for the *associative classifier* has been set as follows:

- **Temporal_threshold**: 60 minutes;
- **Spatial_threshold**: 100 kilometers;
- **Temporal_steps**: 6;
- **Spatial_steps**: 5;
- **Support**: 0;
- **Max_patt_len**: 6;
- **Cities**: San Francisco;

- **Event_types**: Almost_full and increase for the *radius* mode. Almost_full, increase and decrease for the *incoming* one;
- **Neighbor_type**: Radius and incoming;
- **Top_n**: 5;
- **Ignore_set**: Increase for the *radius* mode, Increase and decrease for the *incoming* one;
- **Reproj_mode**: Past;
- **Absolute_mode**: False.

For this experiment, we consider both *full* and *almost_full* as a unique condition. We first train and test the algorithms on the totality of data, and then we do it separately for timeslots of four hours during the whole day. We try different values for the minimum thresholds of *confidence*, *support* and *matching*.

We also compare our results with a **baseline**, which in this case is an *associative classifier* that only makes use of the simple *association rule*:

$$almost\_full\_S0\_T-1 \implies almost\_full\_S0\_T0 \tag{4.5}$$

In other words, if a certain *station* happens to be in an *almost_full* condition (i.e. which represents the situation when the number of docks available is less or equal than a certain threshold), we expect that same *station* to be in an *almost_full* condition even for the next *time window*.
Even if it is a very simple and intuitive *spatio-temporally invariant* pattern, it obtains very good results (among the best), showing that this is a very meaningful pattern.

4.2.1 Neighbor_types: Radius

Here we show the results obtained by setting a **Radius** neighborhood for the *reference station* (i.e. the neighborhood of the *reference station* will consist of all the stations which are placed within **Spatial_steps** concentric circles of radius **Spatial_threshold**).

Associative classifier

In Table 4.1 we see the results in terms of the value of evaluation metrics for the classification task for the Associative classifier and the Baseline. When possible, we also try to limit the list of the selected *association rules* to the ones which have

equal or higher *confidence* with respect to the one associated with the Baseline rule. For the experiments related to the entire daytime, we also try to progressively increase the number of selected rules until we include the Baseline rule.

Table 4.1: Radius mode, Associative classifier, Metrics

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
98.3	98.6	98.2	73.5	99.9	84.2	99.1	6e-3	0.6	6	60	00-03	A	1
98.2	98.1	98.2	73.5	99.9	84.0	99.1	6e-3	0.6	20	60	00-03	A	1
98.3	98.6	98.2	73.5	99.9	84.2	99.1	\	\	0	60	00-03	B	0
92.2	64.6	94.3	45.4	97.3	53.3	95.8	6e-3	0.6	1	60	04-07	A	1
91.2	55.7	94.4	47.1	96.0	51.0	95.2	6e-3	0.6	20	60	04-07	A	1
92.2	64.6	94.3	45.4	97.3	53.3	95.8	\	\	0	60	04-07	B	0
93.7	60.7	95.3	38.0	98.1	46.7	96.7	1e-2	0.6	20	60	08-11	A	1
93.7	58.4	95.8	45.9	97.4	51.4	96.6	\	\	0	60	08-11	B	0
94.2	61.6	95.9	42.9	98.0	50.6	96.9	8e-3	0.6	20	60	12-15	A	1
94.2	61.6	95.9	42.9	98.0	50.6	96.9	\	\	0	60	12-15	B	0
93.3	62.3	95.6	50.8	97.2	56.0	96.4	1e-2	0.6	20	60	16-19	A	1
93.3	62.3	95.6	50.8	97.2	56.0	96.4	\	\	0	60	16-19	B	0
97.4	86.9	98.0	69.0	99.3	76.9	98.6	8e-3	0.6	20	60	20-23	A	1
97.7	91.9	97.9	68.5	99.6	78.5	98.8	\	\	0	60	20-23	B	0
94.5	79.6	95.0	35.0	99.3	48.6	97.1	2e-2	0.6	1	60	all_day	A	1
94.7	76.3	95.5	42.5	98.9	54.6	97.2	2e-2	0.6	2	60	all_day	A	1
94.8	76.2	95.6	43.1	98.9	55.1	97.2	2e-2	0.6	3	60	all_day	A	1
94.9	71.5	96.3	53.7	98.3	61.3	97.3	2e-2	0.6	4	60	all_day	A	1
95.4	71.4	97.1	63.7	97.9	67.4	97.5	2e-2	0.6	5	60	all_day	A	1
95.7	71.3	97.7	71.3	97.7	71.3	97.7	2e-2	0.6	6	60	all_day	A	1
95.0	64.4	97.8	72.4	96.8	68.2	97.3	2e-2	0.6	20	60	all_day	A	1
95.7	71.3	97.7	71.3	97.7	71.3	97.7	\	\	0	60	all_day	B	0

In Table 4.2 we can also see the results in terms of the number of samples which are correctly classified or not in both classes.

Table 4.2: Radius mode, Associative classifier, Confusion matrix

TP	TN	FP	FN	ST	CT	#R	TT	TZ	Alg	MT
1571	31809	22	567	6e-3	0.6	6	60	00-03	A	1
1571	31800	31	567	6e-3	0.6	20	60	00-03	A	1
1571	31809	22	567	\	\	0	60	00-03	B	0
1507	29826	825	1812	6e-3	0.6	1	60	04-07	A	1
1562	29410	1241	1757	6e-3	0.6	20	60	04-07	A	1

1507	29826	825	1812	\	\	0	60	04-07	B	0
933	30874	605	1523	1e-2	0.6	20	60	08-11	A	1
1127	30675	804	1329	\	\	0	60	08-11	B	0
1007	30992	627	1339	8e-3	0.6	20	60	12-15	A	1
1007	30992	627	1339	\	\	0	60	12-15	B	0
1443	30291	873	1398	1e-2	0.6	20	60	16-19	A	1
1443	30291	873	1398	\	\	0	60	16-19	B	0
1456	31604	220	654	8e-3	0.6	20	60	20-23	A	1
1445	31696	128	665	\	\	0	60	20-23	B	0
5325	187207	1361	9885	2e-2	0.6	1	60	all_day	A	1
6464	186563	2005	8746	2e-2	0.6	2	60	all_day	A	1
6560	186521	2047	8650	2e-2	0.6	3	60	all_day	A	1
8169	185304	3264	7041	2e-2	0.6	4	60	all_day	A	1
9692	184693	3875	5518	2e-2	0.6	5	60	all_day	A	1
10840	184212	4356	4370	2e-2	0.6	6	60	all_day	A	1
11017	182485	6083	4193	2e-2	0.6	20	60	all_day	A	1
10840	184212	4356	4370	\	\	0	60	all_day	B	0

As we could expect, given the fact that we are dealing with an unbalanced dataset, we see that the better performances are in general with respect to the *negative* class, which is the majority one. This is caused by the intrinsic nature of the data that we have to classify, since of course the situations during the day in which a *station* is not in a critical condition are considerably higher than the critical ones (especially if we also take the "normal" *time windows* for which no event of interest happens in the *reference station* into consideration). With this data distribution, algorithms are less exposed to samples belonging to the *positive* class, and so they have greater difficulties in classifying correctly this class. In the case of the *associative classifier*, our sequential dataset contains less examples of sequences belonging to the *positive* class, and so it is more difficult to extract patterns of a certain "strength" related to the detection of that class.

We can see that the worst performing metric with respect to the *positive* class is the *recall*, so this typology of classifiers encounters difficulties in assigning the samples belonging to the *positive* class to the correct one, as made evident by the general trend of a higher number of *false negatives* with respect to the *false positives*. In other words, it is more frequent to predict a positive sample to belong to the negative class instead of predicting a negative sample to belong to the positive class. This fact can be explained by the unbalanced data distribution that is the reason why the *sequential data mining* algorithm has more difficulties in extracting *spatio-temporally invariant* patterns useful for the detection of the

positive class: given this situation, it is easier for the classification algorithm to "miss" the *matching* between a sequence and the *association rule* than to wrongly find a match.
So, it is more probable that when a match is "triggered" the algorithm is going to classify a sequence belonging to the *positive* class, since the training data contain less samples belonging to that class and so the associated extracted patterns are more "specific" and less present in the general data population. Anyway, as we can see by the trend followed by the experiments which cover the entire daytime, as we take an increasing amount of *association rules* into consideration, we experience an increasing value of the *recall* metric and a decreasing value of the *precision* one with respect to the positive class. Intuitively, the availability of a higher number of rules allows the classification algorithm to reduce the possibilities of missing the matches between a sequence and a rule, and so to improve the *recall*; at the same time, such a situation affects negatively the *precision* since the classification algorithm takes into account rules of decreasing "strength" in terms of *confidence* and *support*, resulting in more match triggers and so a higher number of *false positives* (i.e. more predictions of positive class associated with sequences that belong to the negative one). This behaviour can be mitigated by raising the threshold of matching: since to "confirm" a prediction associated with the positive class the classification algorithm needs to find multiple matches between a sequence and the rules, the *precision* is increased because the algorithm is "wiser" in its decision of predicting the positive class; on the other hand, the *recall* is decreased because the criteria for a sequence to be classified as belonging to the positive class are stricter.

Subdividing the data into *time zones*, we can see drastic variations in terms of performances.
If we limit the classification task over the night hours (time zones 00-03 and 20-23), we can notice that the algorithm performs better than the other ones, and even with respect to the entire daytime. We can deduce that the *sequential data mining* algorithm is more capable of extracting meaningful *spatio-temporally* invariant patterns from these data, characterized by a more "predictable" bike traffic with respect to the one of the time zones of the morning and the afternoon. We can imagine that during the night the number of users is limited as well as the itineraries of the trips, while during the day there is much more variety of usage of the bike-sharing service.

Looking at the results obtained by the *Baseline* classifier, we see that for the *time zone* data it obtains in general the best results (except for 08-11 and 20-23), showing that the simple pattern of the permanence of the critical condition from a *time window* to the next one is simple but effective and among the most "reliable" ones.

Other classifiers

In Table 4.3 we see the results in terms of the value of evaluation metrics for the classification task for the Decision tree and the Random forest. For each *time zone*, we report the results obtained by the algorithms with the default configuration of *hyperparameters* and with the configurations optimized over the evaluation metrics mentioned in Paragraph 4.1.2 by means of the *validation* process illustrated in Paragraph 4.1.3.

Table 4.3: Radius mode, Other classifiers, Metrics

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	F	P	0
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	F	F1	0
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	F	\	0
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	T	P	0
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	T	F1	0
98.3	98.7	98.2	73.3	99.9	84.1	99.1	0	60	00-03	T	\	0
91.5	63.0	93.0	31.4	98.0	41.9	95.4	0	60	04-07	F	P	0
92.0	63.7	93.9	41.1	97.5	50.0	95.6	0	60	04-07	F	F1	0
92.0	63.7	93.9	41.0	97.5	49.9	95.6	0	60	04-07	F	\	0
92.2	64.6	94.3	45.4	97.3	53.3	95.8	0	60	04-07	T	P	0
92.2	64.6	94.3	45.4	97.3	53.3	95.8	0	60	04-07	T	F1	0
91.9	62.7	93.8	41.0	97.4	49.6	95.6	0	60	04-07	T	\	0
92.8	42.9	92.8	0.4	99.9	0.7	96.2	0	60	08-11	F	P	0
93.6	60.6	94.9	31.9	98.4	41.8	96.6	0	60	08-11	F	F1	0
93.4	58.3	94.8	31.3	98.3	40.7	96.5	0	60	08-11	F	\	0
93.7	58.6	95.8	45.4	97.5	51.2	96.6	0	60	08-11	T	P	0
93.7	58.4	95.8	45.9	97.4	51.4	96.6	0	60	08-11	T	F1	0
93.2	55.2	94.8	31.5	98.0	40.1	96.4	0	60	08-11	T	\	0
93.4	67.7	93.7	9.0	99.7	15.8	96.6	0	60	12-15	F	P	0
94.0	60.9	95.5	37.7	98.2	46.6	96.8	0	60	12-15	F	F1	0
93.9	60.3	95.4	35.8	98.3	44.9	96.8	0	60	12-15	F	\	0
94.2	61.6	95.9	42.9	98.0	50.6	96.9	0	60	12-15	T	P	0
94.2	61.6	95.9	42.9	98.0	50.6	96.9	0	60	12-15	T	F1	0
93.7	58.3	95.2	33.4	98.2	42.5	96.7	0	60	12-15	T	\	0
92.0	68.6	92.2	7.1	99.7	12.8	95.8	0	60	16-19	F	P	0
93.0	62.1	94.8	41.3	97.7	49.6	96.2	0	60	16-19	F	F1	0
93.0	61.9	94.8	40.8	97.7	49.2	96.2	0	60	16-19	F	\	0
92.9	61.4	94.7	39.4	97.7	48.0	96.2	0	60	16-19	T	P	0
93.3	62.3	95.6	50.8	97.2	56.0	96.4	0	60	16-19	T	F1	0

92.7	60.3	94.6	38.3	97.7	46.9	96.1	0	60	16-19	T	\	0
96.9	92.8	97.0	54.0	99.7	68.3	98.4	0	60	20-23	F	P	0
97.6	92.1	97.8	66.4	99.6	77.2	98.7	0	60	20-23	F	F1	0
97.5	91.8	97.7	65.4	99.6	76.4	98.7	0	60	20-23	F	\	0
97.7	91.9	97.9	68.5	99.6	78.5	98.8	0	60	20-23	T	P	0
97.7	91.9	97.9	68.5	99.6	78.5	98.8	0	60	20-23	T	F1	0
97.4	91.5	97.6	63.5	99.6	75.0	98.6	0	60	20-23	T	\	0
93.0	83.9	93.0	7.4	99.9	13.6	96.3	0	60	all_day	F	P	0
95.7	73.7	97.3	65.8	98.1	69.5	97.7	0	60	all_day	F	F1	0
95.4	72.4	97.0	62.9	98.1	67.3	97.6	0	60	all_day	F	\	0
95.7	71.3	97.7	71.3	97.7	71.3	97.7	0	60	all_day	T	P	0
95.7	71.3	97.7	71.3	97.7	71.3	97.7	0	60	all_day	T	F1	0
94.4	64.3	96.6	56.9	97.4	60.4	97.0	0	60	all_day	T	\	0

In Table 4.4 we can also see the results in terms of the number of samples which are correctly classified or not in both classes.

Table 4.4: Radius mode, Other classifiers, Confusion matrix

TP	TN	FP	FN	#R	TT	TZ	Alg	Val	MT
1571	31809	22	567	0	60	00-03	F	P	0
1571	31809	22	567	0	60	00-03	F	F1	0
1571	31809	22	567	0	60	00-03	F	\	0
1571	31809	22	567	0	60	00-03	T	P	0
1571	31809	22	567	0	60	00-03	T	F1	0
1568	31810	21	570	0	60	00-03	T	\	0
1043	30039	612	2276	0	60	04-07	F	P	0
1364	29874	777	1955	0	60	04-07	F	F1	0
1362	29874	777	1957	0	60	04-07	F	\	0
1507	29826	825	1812	0	60	04-07	T	P	0
1507	29826	825	1812	0	60	04-07	T	F1	0
1361	29841	810	1958	0	60	04-07	T	\	0
9	31467	12	2447	0	60	08-11	F	P	0
784	30970	509	1672	0	60	08-11	F	F1	0
768	30930	549	1688	0	60	08-11	F	\	0
1115	30691	788	1341	0	60	08-11	T	P	0
1127	30675	804	1329	0	60	08-11	T	F1	0
773	30851	628	1683	0	60	08-11	T	\	0
210	31519	100	2136	0	60	12-15	F	P	0
884	31052	567	1462	0	60	12-15	F	F1	0

840	31066	553	1506	0	60	12-15	F	\	0
1007	30992	627	1339	0	60	12-15	T	P	0
1007	30992	627	1339	0	60	12-15	T	F1	0
783	31059	560	1563	0	60	12-15	T	\	0
201	31072	92	2640	0	60	16-19	F	P	0
1174	30447	717	1667	0	60	16-19	F	F1	0
1160	30450	714	1681	0	60	16-19	F	\	0
1120	30461	703	1721	0	60	16-19	T	P	0
1443	30291	873	1398	0	60	16-19	T	F1	0
1089	30446	718	1752	0	60	16-19	T	\	0
1140	31736	88	970	0	60	20-23	F	P	0
1402	31704	120	708	0	60	20-23	F	F1	0
1379	31701	123	731	0	60	20-23	F	\	0
1445	31696	128	665	0	60	20-23	T	P	0
1445	31696	128	665	0	60	20-23	T	F1	0
1340	31700	124	770	0	60	20-23	T	\	0
1128	188351	217	14082	0	60	all_day	F	P	0
10014	184994	3574	5196	0	60	all_day	F	F1	0
9569	184927	3641	5641	0	60	all_day	F	\	0
10840	184212	4356	4370	0	60	all_day	T	P	0
10840	184212	4356	4370	0	60	all_day	T	F1	0
8657	183752	4816	6553	0	60	all_day	T	\	0

As we could expect, given the fact that we are dealing with an unbalanced dataset, we see that the better performances are in general with respect to the *negative* class, which is the majority one, also deploying these more traditional classifiers. In the case of the *Decision tree* and the *Random forest*, with this data distribution the algorithms are less exposed to samples belonging to the *positive* class, and so they have greater difficulties in classifying correctly this class. Specifically, these algorithms encounter difficulties with the construction of paths inside the tree diagrams which end with leaves that allow for the prediction of the minority class: indeed, it is hard for the algorithm to find the distinctive features that allow to "isolate" these samples inside the feature space where they lie.

We can see that the worst performing metric with respect to the *positive* class is again the *recall*, so also this typology of classifiers encounters difficulties in assigning the samples belonging to the *positive* class to the correct one, as made evident by the general trend of a higher number of *false negatives* with respect to the *false positives*. This fact can be explained by the unbalanced data distribution that is the same reason why the *sequential data mining* algorithm has more difficulties

in extracting *spatio-temporally invariant* patterns useful for the detection of the positive class (the data representation is a binary version of the sequences processed by the *associative classifier*). In a similar fashion to what happens with the associative classifier, since we have less paths inside the tree diagrams which lead to a prediction of positive class, it is easier for the classification algorithm to "miss" the detection of a sample belonging to the positive class than to wrongly assign a prediction of positive class. So, it is more probable that when a sample walks a path which ends with a leaf predicting the positive class the algorithm is going to classify a sequence actually belonging to the positive class, since the training data contain less samples belonging to that class and so the associated extracted traits contained along the path are more "specific" and less present in the general data population.

Subdividing the data into *time zones*, we can see drastic variations in terms of performances.
If we look at the 08-11, 12-15, 16-19 time zones and at the whole daytime, we can notice that the *Random forest* set with the configuration of *hyperparameters* optimized over the *precision* metric gets very low results in terms of the *recall* metric with respect to the positive class. In these situations, we see that the achievement of better performances in terms of *precision* is correlated with very bad performances in terms of *recall*: the algorithm is "wiser" in its predictions but it misses the detection of a lot of samples belonging to the positive class. This phenomenon is not as much present in the *Decision tree* algorithm, showing to us that optimizing *precision* in a single tree diagram has a negligible impact on the *recall*, whereas doing it for an ensemble exacerbates the gap between the two metrics. Intuitively, spreading this optimization over multiple diagram trees causes an increasing distance between the two metrics since the compromise between the two is iterated many times.
If we limit the classification task over the night hours (time zones 00-03 and 20-23), we can notice that the algorithm performs better than the other ones, and even with respect to the entire daytime. As we have noticed by means of the experiments related to the *associative classifier*, these time zones are characterized by a more "predictable" bike traffic with respect to the one of the time zones of the morning and the afternoon, and so that is reflected in the obtained results.

00-03

We report the best results obtained in the 00-03 time zone, according to the *precision* metric with respect to the positive class.

Table 4.5: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
98.3	98.6	98.2	73.5	99.9	84.2	99.1	6e-3	0.6	6	60	00-03	A	1
98.3	98.6	98.2	73.5	99.9	84.2	99.1	\	\	0	60	00-03	B	0

Table 4.6: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
98.3	98.7	98.2	73.3	99.9	84.1	99.1	0	60	00-03	T	\	0

For this time zone, we see that perform best the *associative classifier* keeping just the rules with *confidence* equal or higher than the *baseline* (together with the baseline model) and the *decision tree* with the default configuration of hyperparameters. We observe that the performances are very similar, with a slight prevalence of the Decision tree for what concerns the *precision* and of the Associative classifier for what concerns the *recall*.

04-07

We report the best results obtained in the 04-07 time zone, according to the *precision* metric with respect to the positive class.

Table 4.7: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
92.2	64.6	94.3	45.4	97.3	53.3	95.8	\	\	0	60	04-07	B	0

Table 4.8: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
92.2	64.6	94.3	45.4	97.3	53.3	95.8	0	60	04-07	T	P	0
92.2	64.6	94.3	45.4	97.3	53.3	95.8	0	60	04-07	T	F1	0

For this time zone, we see that performs best the *baseline* model and the *decision tree* with configuration of hyperparameters optimized towards *precision* and *F1 score*. We observe that all the reported models obtain almost equal performances.

08-11

We report the best results obtained in the 08-11 time zone, according to the *precision* metric with respect to the positive class.

Table 4.9: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
93.7	60.7	95.3	38.0	98.1	46.7	96.7	1e-2	0.6	20	60	08-11	A	1

Table 4.10: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
93.6	60.6	94.9	31.9	98.4	41.8	96.6	0	60	08-11	F	F1	0

For this time zone, we see that perform best the *associative classifier* and the *random forest* with configuration of hyperparameters optimized towards the *F1 score*. We observe that the performances are very similar for what concerns the *precision* (with a slight prevalence of the Associative classifier), while there is a larger gap for what concerns the *recall* (still in favour of the Associative classifier).

12-15

We report the best results obtained in the 12-15 time zone, according to the *precision* metric with respect to the positive class.

Table 4.11: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
94.2	61.6	95.9	42.9	98.0	50.6	96.9	8e-3	0.6	20	60	12-15	A	1
94.2	61.6	95.9	42.9	98.0	50.6	96.9	\	\	0	60	12-15	B	0

Table 4.12: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
93.4	67.7	93.7	9.0	99.7	15.8	96.6	0	60	12-15	F	P	0

For this time zone, we see that perform best the *associative classifier* together with the *baseline* model and the *random forest* with configuration of hyperparameters optimized towards the *precision*. We observe that the performances have a bit of gap for what concerns the *precision* (with a prevalence of the Random forest), while there is a huge gap for what concerns the *recall* (instead in favour of the Associative classifier, together with the Baseline model).

16-19

We report the best results obtained in the 16-19 time zone, according to the *precision* metric with respect to the positive class.

Table 4.13: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
93.3	62.3	95.6	50.8	97.2	56.0	96.4	1e-2	0.6	20	60	16-19	A	1
93.3	62.3	95.6	50.8	97.2	56.0	96.4	\	\	0	60	16-19	B	0

Table 4.14: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
92.0	68.6	92.2	7.1	99.7	12.8	95.8	0	60	16-19	F	P	0

For this time zone, we see that perform best the *associative classifier* together with the *baseline* model and the *random forest* with configuration of hyperparameters optimized towards the *precision*. Like the 12-15 time zone, we observe that the performances have a bit of gap for what concerns the *precision* (with a prevalence of the Random forest), while there is a huge gap for what concerns the *recall* (instead in favour of the Associative classifier, together with the Baseline model).

20-23

We report the best results obtained in the 20-23 time zone, according to the *precision* metric with respect to the positive class.

Table 4.15: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
97.7	91.9	97.9	68.5	99.6	78.5	98.8	\	\	0	60	20-23	B	0

Table 4.16: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
96.9	92.8	97.0	54.0	99.7	68.3	98.4	0	60	20-23	F	P	0

For this time zone, we see that perform best the *baseline* model and the *random forest* with configuration of hyperparameters optimized towards the *precision*. We observe that the performances are similar for what concerns the *precision* (with a prevalence of the Random forest), while there is a bit of gap for what concerns the *recall* (instead in favour of the the Baseline model).

All day

We report the best results obtained in the whole daytime, according to the *precision* metric with respect to the positive class.

Table 4.17: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
94.5	79.6	95.0	35.0	99.3	48.6	97.1	2e-2	0.6	1	60	all_day	A	1

Table 4.18: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
93.0	83.9	93.0	7.4	99.9	13.6	96.3	0	60	all_day	F	P	0

For this time zone, we see that perform best the *associative classifier* keeping just the top rule in terms of *confidence* and the *random forest* with configuration of hyperparameters optimized towards the *precision*. We observe that the performances are a bit different for what concerns the *precision* (with a prevalence of the Random forest), while there is a huge gap for what concerns the *recall* (instead in favour of the the Associative classifier).

4.2.2 Neighbor_types: Incoming

Here we show the results obtained by setting an **Incoming** neighborhood for the *reference station* (i.e. the neighborhood of the *reference station* will consist of the **Top_n** stations which have the highest number of records into the *trip* dataset).

Associative classifier

In Table 4.19 we see the results in terms of the value of evaluation metrics for the classification task for the Associative classifier and the Baseline. When possible, we also try to limit the list of the selected *association rules* to the ones which have equal or higher *confidence* with respect to the one associated with the Baseline rule. For the experiments related to the entire daytime, we also try to progressively increase the number of selected rules until we include the Baseline rule.

Table 4.19: Incoming mode, Associative classifier, Metrics

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
98.3	98.6	98.2	73.5	99.9	84.2	99.1	6e-3	0.6	7	60	00-03	A	1
98.2	98.1	98.2	73.5	99.9	84.0	99.1	6e-3	0.6	20	60	00-03	A	1

98.3	98.6	98.2	73.5	99.9	84.2	99.1	\	\	0	60	00-03	B	0
92.2	64.6	94.3	45.4	97.3	53.3	95.8	7e-3	0.6	1	60	04-07	A	1
91.2	55.7	94.4	47.1	96.0	51.0	95.2	7e-3	0.6	20	60	04-07	A	1
92.2	64.6	94.3	45.4	97.3	53.3	95.8	\	\	0	60	04-07	B	0
93.6	59.7	95.2	36.6	98.1	45.3	96.6	8e-3	0.6	20	60	08-11	A	1
93.7	58.4	95.8	45.9	97.4	51.4	96.6	\	\	0	60	08-11	B	0
94.2	61.6	95.9	42.9	98.0	50.6	96.9	8e-3	0.6	20	60	12-15	A	1
94.2	61.6	95.9	42.9	98.0	50.6	96.9	\	\	0	60	12-15	B	0
93.3	62.3	95.6	50.8	97.2	56.0	96.4	1e-2	0.6	20	60	16-19	A	1
93.3	62.3	95.6	50.8	97.2	56.0	96.4	\	\	0	60	16-19	B	0
97.4	86.9	98.0	69.0	99.3	76.9	98.6	1e-2	0.6	20	60	20-23	A	1
97.7	91.9	97.9	68.5	99.6	78.5	98.8	\	\	0	60	20-23	B	0
94.5	79.6	95.0	35.0	99.3	48.6	97.1	2e-2	0.6	1	60	all_day	A	1
94.7	76.3	95.5	42.5	98.9	54.6	97.2	2e-2	0.6	2	60	all_day	A	1
94.8	76.2	95.6	43.1	98.9	55.1	97.2	2e-2	0.6	3	60	all_day	A	1
94.9	71.5	96.3	53.7	98.3	61.3	97.3	2e-2	0.6	4	60	all_day	A	1
95.4	71.4	97.1	63.7	97.9	67.4	97.5	2e-2	0.6	5	60	all_day	A	1
95.7	71.3	97.7	71.3	97.7	71.3	97.7	2e-2	0.6	6	60	all_day	A	1
95.0	64.4	97.8	72.4	96.8	68.2	97.3	2e-2	0.6	20	60	all_day	A	1
95.7	71.3	97.7	71.3	97.7	71.3	97.7	\	\	0	60	all_day	B	0

In Table 4.20 we can also see the results in terms of the number of samples which are correctly classified or not in both classes.

Table 4.20: Incoming mode, Associative classifier, Confusion matrix

TP	TN	FP	FN	ST	CT	#R	TT	TZ	Alg	MT
1571	31809	22	567	6e-3	0.6	7	60	00-03	A	1
1571	31800	31	567	6e-3	0.6	20	60	00-03	A	1
1571	31809	22	567	\	\	0	60	00-03	B	0
1507	29826	825	1812	7e-3	0.6	1	60	04-07	A	1
1562	29410	1241	1757	7e-3	0.6	20	60	04-07	A	1
1507	29826	825	1812	\	\	0	60	04-07	B	0
898	30872	607	1558	8e-3	0.6	20	60	08-11	A	1
1127	30675	804	1329	\	\	0	60	08-11	B	0
1007	30992	627	1339	8e-3	0.6	20	60	12-15	A	1
1007	30992	627	1339	\	\	0	60	12-15	B	0
1443	30291	873	1398	1e-2	0.6	20	60	16-19	A	1
1443	30291	873	1398	\	\	0	60	16-19	B	0

1456	31604	220	654	1e-2	0.6	20	60	20-23	A	1
1445	31696	128	665	\	\	0	60	20-23	B	0
5325	187207	1361	9885	2e-2	0.6	1	60	all_day	A	1
6464	186563	2005	8746	2e-2	0.6	2	60	all_day	A	1
6560	186521	2047	8650	2e-2	0.6	3	60	all_day	A	1
8169	185304	3264	7041	2e-2	0.6	4	60	all_day	A	1
9692	184693	3875	5518	2e-2	0.6	5	60	all_day	A	1
10840	184212	4356	4370	2e-2	0.6	6	60	all_day	A	1
11017	182485	6083	4193	2e-2	0.6	20	60	all_day	A	1
10840	184212	4356	4370	\	\	0	60	all_day	B	0

We can make observations similar to what we have seen in the experiments related to the *radius* mode.
Setting this amount of *association rules*, we end up selecting the same *spatio-temporally invariant* patterns, which are associated with the *reference station*: we can then deduce that this typology of patterns are the most robust in both modalities. Thinking about the neighborhood composition of this modality, there is a wide range of distances from the *reference station* at which the neighbors are located, resulting in an increased difficulty for the *sequential data mining* algorithm to extract patterns related to the neighbors.

Other classifiers

In Table 4.21 we see the results in terms of the value of evaluation metrics for the classification task for the Decision tree and the Random forest. For each *time zone*, we report the results obtained by the algorithms with the default configuration of *hyperparameters* and with the configurations optimized over the evaluation metrics mentioned in Paragraph 4.1.2 by means of the *validation* process illustrated in Paragraph 4.1.3.

Table 4.21: Incoming mode, Other classifiers, Metrics

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	F	P	0
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	F	F1	0
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	F	\	0
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	T	P	0
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	T	F1	0
98.3	98.6	98.2	73.3	99.9	84.1	99.1	0	60	00-03	T	\	0
91.7	63.6	93.3	34.9	97.8	45.1	95.5	0	60	04-07	F	P	0

92.4	66.5	94.1	44.0	97.6	53.0	95.8	0	60	04-07	F	F1	0
92.3	65.8	94.2	44.1	97.5	52.8	95.8	0	60	04-07	F	\	0
92.2	64.6	94.3	45.4	97.3	53.3	95.8	0	60	04-07	T	P	0
92.2	64.6	94.3	45.4	97.3	53.3	95.8	0	60	04-07	T	F1	0
92.3	66.3	94.1	44.0	97.6	52.9	95.8	0	60	04-07	T	\	0
93.7	61.0	95.1	35.0	98.3	44.5	96.6	0	60	08-11	F	P	0
93.6	60.7	94.9	32.5	98.4	42.3	96.6	0	60	08-11	F	F1	0
93.7	61.0	95.1	35.0	98.3	44.5	96.6	0	60	08-11	F	\	0
93.7	60.1	95.3	38.3	98.0	46.8	96.6	0	60	08-11	T	P	0
93.7	58.4	95.8	45.9	97.4	51.4	96.6	0	60	08-11	T	F1	0
93.7	61.0	95.1	35.0	98.3	44.5	96.6	0	60	08-11	T	\	0
94.1	63.2	95.3	34.6	98.5	44.7	96.9	0	60	12-15	F	P	0
94.2	61.5	95.8	42.6	98.0	50.3	96.9	0	60	12-15	F	F1	0
94.2	61.5	95.8	42.7	98.0	50.4	96.9	0	60	12-15	F	\	0
94.2	62.0	95.8	42.3	98.1	50.3	96.9	0	60	12-15	T	P	0
94.2	61.6	95.9	42.9	98.0	50.6	96.9	0	60	12-15	T	F1	0
94.2	62.0	95.8	42.3	98.1	50.3	96.9	0	60	12-15	T	\	0
92.7	66.9	93.5	24.5	98.9	35.9	96.1	0	60	16-19	F	P	0
93.2	64.4	94.7	40.4	98.0	49.7	96.3	0	60	16-19	F	F1	0
93.4	63.3	95.4	48.5	97.4	54.9	96.4	0	60	16-19	F	\	0
93.4	63.8	95.3	47.6	97.5	54.5	96.4	0	60	16-19	T	P	0
93.3	64.0	95.2	45.4	97.7	53.1	96.4	0	60	16-19	T	F1	0
93.4	63.8	95.3	47.6	97.5	54.5	96.4	0	60	16-19	T	\	0
97.7	91.9	97.9	68.4	99.6	78.4	98.8	0	60	20-23	F	P	0
97.7	91.9	97.9	68.4	99.6	78.4	98.8	0	60	20-23	F	F1	0
97.7	91.9	97.9	68.4	99.6	78.4	98.8	0	60	20-23	F	\	0
97.7	91.9	97.9	68.5	99.6	78.5	98.8	0	60	20-23	T	P	0
97.7	91.9	97.9	68.5	99.6	78.5	98.8	0	60	20-23	T	F1	0
97.7	91.9	97.9	68.4	99.6	78.4	98.8	0	60	20-23	T	\	0
95.7	72.4	97.5	68.5	97.9	70.4	97.7	0	60	all_day	F	P	0
95.7	71.3	97.7	71.2	97.7	71.3	97.7	0	60	all_day	F	F1	0
95.7	72.4	97.5	68.4	97.9	70.3	97.7	0	60	all_day	F	\	0
95.7	72.3	97.4	68.1	97.9	70.2	97.7	0	60	all_day	T	P	0
95.7	71.3	97.7	71.3	97.7	71.3	97.7	0	60	all_day	T	F1	0
95.7	72.3	97.4	68.1	97.9	70.1	97.7	0	60	all_day	T	\	0

In Table 4.22 we can also see the results in terms of the number of samples which are correctly classified or not in both classes.

Table 4.22: Incoming mode, Other classifiers, Confusion matrix

TP	TN	FP	FN	#R	TT	TZ	Alg	Val	MT
1571	31809	22	567	0	60	00-03	F	P	0
1571	31809	22	567	0	60	00-03	F	F1	0
1571	31809	22	567	0	60	00-03	F	\	0
1571	31809	22	567	0	60	00-03	T	P	0
1571	31809	22	567	0	60	00-03	T	F1	0
1568	31809	22	570	0	60	00-03	T	\	0
1159	29987	664	2160	0	60	04-07	F	P	0
1460	29916	735	1859	0	60	04-07	F	F1	0
1465	29890	761	1854	0	60	04-07	F	\	0
1507	29826	825	1812	0	60	04-07	T	P	0
1507	29826	825	1812	0	60	04-07	T	F1	0
1460	29909	742	1859	0	60	04-07	T	\	0
859	30929	550	1597	0	60	08-11	F	P	0
798	30963	516	1658	0	60	08-11	F	F1	0
859	30929	550	1597	0	60	08-11	F	\	0
940	30855	624	1516	0	60	08-11	T	P	0
1127	30675	804	1329	0	60	08-11	T	F1	0
859	30929	550	1597	0	60	08-11	T	\	0
811	31146	473	1535	0	60	12-15	F	P	0
999	30993	626	1347	0	60	12-15	F	F1	0
1001	30993	626	1345	0	60	12-15	F	\	0
993	31011	608	1353	0	60	12-15	T	P	0
1007	30992	627	1339	0	60	12-15	T	F1	0
993	31011	608	1353	0	60	12-15	T	\	0
696	30819	345	2145	0	60	16-19	F	P	0
1149	30530	634	1692	0	60	16-19	F	F1	0
1377	30367	797	1464	0	60	16-19	F	\	0
1351	30397	767	1490	0	60	16-19	T	P	0
1291	30437	727	1550	0	60	16-19	T	F1	0
1351	30397	767	1490	0	60	16-19	T	\	0
1443	31696	128	667	0	60	20-23	F	P	0
1443	31696	128	667	0	60	20-23	F	F1	0
1443	31696	128	667	0	60	20-23	F	\	0
1445	31696	128	665	0	60	20-23	T	P	0
1445	31696	128	665	0	60	20-23	T	F1	0
1443	31696	128	667	0	60	20-23	T	\	0
10415	184591	3977	4795	0	60	all_day	F	P	0

10832	184215	4353	4378	0	60	all_day	F	F1	0
10402	184597	3971	4808	0	60	all_day	F	\	0
10364	184595	3973	4846	0	60	all_day	T	P	0
10840	184212	4356	4370	0	60	all_day	T	F1	0
10358	184592	3976	4852	0	60	all_day	T	\	0

As for the *radius* modality, given the fact that we are dealing with an unbalanced dataset, we see that the better performances are in general with respect to the *negative* class, which is the majority one, also deploying these more traditional classifiers.

We can see that the worst performing metric with respect to the *positive* class is again the *recall*, so also in this modality this typology of classifiers encounters difficulties in assigning the samples belonging to the *positive* class to the correct one, as made evident by the general trend of a higher number of *false negatives* with respect to the *false positives*.

Subdividing the data into *time zones*, we can see drastic variations in terms of performances.
If we look at the 08-11, 12-15, 16-19 time zones and at the whole daytime, we can notice now that the *Random forest* set with the configuration of *hyperparameters* optimized over the *precision* metric does not get anymore very low results in terms of the *recall* metric with respect to the positive class, as opposed to the situation in the *radius* modality, so it seems that the gap between *precision* and *recall* is not as much exacerbated in this modality.
If we limit the classification task over the night hours (time zones 00-03 and 20-23), we can again notice that the algorithm performs better than the other ones, and even with respect to the entire daytime.

Comparing the two modalities, we can notice slightly better performances of the *incoming* modality with respect to the *radius* one, and this could be caused by a more effective data representation for this modality.
Indeed, for the *radius* modality we have a column for each combination of *event*, *station* and *time window*; instead, for the *incoming* modality, we just have a single column which represent the whole neighborhood for each *time window*, signalling if at least one of the stations belonging to the neighborhood of the *reference station* is in a *decrease* state. In this way, we limit the number of features as opposed to the phenomenon of the *curse of dimensionality*, since usually a higher number of features leads to more complex models which fail to generalize to data not seen during the training process. The single column to represent the whole neighborhood

within a certain *time window* seems to be a more distinctive feature with respect to represent the condition of each neighbor separately.

00-03

We report the best results obtained in the 00-03 time zone, according to the *precision* metric with respect to the positive class.

Table 4.23: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
98.3	98.6	98.2	73.5	99.9	84.2	99.1	6e-3	0.6	6	60	00-03	A	1
98.3	98.6	98.2	73.5	99.9	84.2	99.1	\	\	0	60	00-03	B	0

Table 4.24: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	F	P	0
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	F	F1	0
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	F	\	0
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	T	P	0
98.3	98.6	98.2	73.5	99.9	84.2	99.1	0	60	00-03	T	F1	0
98.3	98.6	98.2	73.3	99.9	84.1	99.1	0	60	00-03	T	\	0

For this time zone, we see that perform best the *associative classifier* keeping just the rules with *confidence* equal or higher than the *baseline* (together with the baseline model) and all the configurations of *decision tree* and *random forest*. We observe that the performances are very similar, with a slightly worse performance of the default Decision tree for what concerns the *recall*.

04-07

We report the best results obtained in the 04-07 time zone, according to the *precision* metric with respect to the positive class.

Table 4.25: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
92.2	64.6	94.3	45.4	97.3	53.3	95.8	\	\	0	60	04-07	B	0

Table 4.26: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
92.4	66.5	94.1	44.0	97.6	53.0	95.8	0	60	04-07	F	F1	0

For this time zone, we see that performs best the *baseline* model and the *random forest* with configuration of hyperparameters optimized towards the *F1 score*. We observe similar performances, with a slight prevalence of Random forest for what concerns the *precision* and of the Baseline in terms of *recall*.

08-11

We report the best results obtained in the 08-11 time zone, according to the *precision* metric with respect to the positive class.

Table 4.27: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
93.7	60.7	95.3	38.0	98.1	46.7	96.7	1e-2	0.6	20	60	08-11	A	1

Table 4.28: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
93.7	61.0	95.1	35.0	98.3	44.5	96.6	0	60	08-11	F	P	0
93.7	61.0	95.1	35.0	98.3	44.5	96.6	0	60	08-11	F	\	0
93.7	61.0	95.1	35.0	98.3	44.5	96.6	0	60	08-11	T	\	0

For this time zone, we see that perform best the *associative classifier* and the *random forest* with configuration of hyperparameters optimized towards the *precision* (together with the default configurations of Random forest and Decision tree). We observe that the performances are very similar for what concerns the *precision* (with a slight prevalence of the traditional classifier), while there is a larger gap for what concerns the *recall* (instead in favour of the Associative classifier).

12-15

We report the best results obtained in the 12-15 time zone, according to the *precision* metric with respect to the positive class.

Table 4.29: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT

94.2	61.6	95.9	42.9	98.0	50.6	96.9	8e-3	0.6	20	60	12-15	A	1
94.2	61.6	95.9	42.9	98.0	50.6	96.9	\	\	0	60	12-15	B	0

Table 4.30: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
94.1	63.2	95.3	34.6	98.5	44.7	96.9	0	60	12-15	F	P	0

For this time zone, we see that perform best the *associative classifier* together with the *baseline* model and the *random forest* with configuration of hyperparameters optimized towards the *precision*. We observe that the performances have a bit of gap for what concerns the *precision* (with a prevalence of the Random forest), while there is a larger gap for what concerns the *recall* (instead in favour of the Associative classifier, together with the Baseline model).

16-19

We report the best results obtained in the 16-19 time zone, according to the *precision* metric with respect to the positive class.

Table 4.31: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
93.3	62.3	95.6	50.8	97.2	56.0	96.4	1e-2	0.6	20	60	16-19	A	1
93.3	62.3	95.6	50.8	97.2	56.0	96.4	\	\	0	60	16-19	B	0

Table 4.32: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
92.7	66.9	93.5	24.5	98.9	35.9	96.1	0	60	16-19	F	P	0

For this time zone, we see that perform best the *associative classifier* together with the *baseline* model and the *random forest* with configuration of hyperparameters optimized towards the *precision*. Like the 12-15 time zone, we observe that the performances have a bit of gap for what concerns the *precision* (with a prevalence of the Random forest), while there is a larger gap for what concerns the *recall* (instead in favour of the Associative classifier, together with the Baseline model).

20-23

We report the best results obtained in the 20-23 time zone, according to the *precision* metric with respect to the positive class.

Table 4.33: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
97.7	91.9	97.9	68.5	99.6	78.5	98.8	\	\	0	60	20-23	B	0

Table 4.34: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
97.7	91.9	97.9	68.4	99.6	78.4	98.8	0	60	20-23	F	P	0
97.7	91.9	97.9	68.4	99.6	78.4	98.8	0	60	20-23	F	F1	0
97.7	91.9	97.9	68.4	99.6	78.4	98.8	0	60	20-23	F	\	0
97.7	91.9	97.9	68.5	99.6	78.5	98.8	0	60	20-23	T	P	0
97.7	91.9	97.9	68.5	99.6	78.5	98.8	0	60	20-23	T	F1	0
97.7	91.9	97.9	68.4	99.6	78.4	98.8	0	60	20-23	T	\	0

For this time zone, we see that perform best the *baseline* model and all the configurations of the other classifiers. We observe that the performances are similar between the two model categories.

All day

We report the best results obtained in the whole daytime, according to the *precision* metric with respect to the positive class.

Table 4.35: Radius mode, Associative classifier, Best

A	P+	P-	R+	R-	F+	F-	ST	CT	#R	TT	TZ	Alg	MT
94.5	79.6	95.0	35.0	99.3	48.6	97.1	2e-2	0.6	1	60	all_day	A	1

Table 4.36: Radius mode, Other classifiers, Best

A	P+	P-	R+	R-	F+	F-	#R	TT	TZ	Alg	Val	MT
95.7	72.4	97.5	68.5	97.9	70.4	97.7	0	60	all_day	F	P	0
95.7	72.4	97.5	68.4	97.9	70.3	97.7	0	60	all_day	F	\	0

For this time zone, we see that perform best the *associative classifier* keeping just the top rule in terms of *confidence* and the *random forest* with configuration

of hyperparameters optimized towards the *precision* (together with the default configuration). We observe that the performances are a bit different for what concerns the *precision* (with a prevalence of the Associative classifier), while there is a huge gap for what concerns the *recall* (instead in favour of the the Random forest).

Chapter 5

Conclusions

In this thesis, we have explored what it takes to deploy an *associative classifier* to classify spatio-temporal data, and we have compared it with other more traditional classifiers, as the *Decision tree* and the *Random forest*.
First, we have to define the classes that have to be detected by the classification algorithm, which identify some specific condition that emerges from the data (e.g. the saturation of the docks in a bike station in a certain timestamp). Then, the data coming from a dataset of logfile format have to be transformed into a *sequential* format, in order to then apply a *sequential data mining* algorithm to extract *spatio-temporally invariant* patterns that can be interpreted as *association rules* (necessary for the classification algorithm in order to perform the predictions). What distinguishes this type of classifier from the other ones is the remarkable *interpretability* of the predictions made by it. Indeed, the decisional process of the classification algorithm is more transparent since we can directly look at the patterns extracted by the mining algorithm to understand what are the co-occurrences of *events* in the past that could determine the happening of a critical condition in a future period of time.

As we have seen, it is difficult to conciliate satisfactory values of the metrics *precision* and *recall* for the minority class: for this reason is fundamental to define which metric is more important to be optimized for our purposes.
With a poor *precision*, we end up having a lot of *false positives*: in our case study, that implies assuming that a station has filled all its docks, and so it is ready to be deprived of some bikes to be destined to other stations that need them, even if actually is not. With a poor *recall*, we end up having a lot of *false negatives*: in our case study, that implies assuming that a station has not filled all its docks, and so it is ready to host incoming bikes, even if actually is not.
A choice can be made only by means of an accurate analysis by the decision-makers, whose can numerically evaluate the cost and benefits of these scenarios

and identify the most suitable option, even with differentiated solutions according to the geographical location of the single stations.
What emerges from our experiments is that in general the traditional classifiers perform better for what concerns the *precision* (in particular the *random forest*), while the *associative classifier* for what concerns the *recall*.

As further developments of the work of this thesis, we can consider to test other configuration of parameters for the algorithm which generates the sequences in input to *Prefixspan*, as well as different subdivision of the data into *time zones*.
Moreover, we can try to limit the negative effects of unbalanced data distribution, taking into account some techniques of *oversampling* to increase the number of samples belonging to the minority class to which the classification algorithm is exposed during the training process.
In addition to that, we can also try the application of the *associative classifier* to other kind of spatio-temporal data, extracting from them *spatio-temporally invariant* patterns after the appropriate preprocessing stage of data transformation.

Bibliography

[1] Steinbach Tan and Kumar. *Introduction to Data Mining*. McGraw Hill, 2006 (cit. on p. 3).

[2] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014 (cit. on p. 3).

[3] Jian Pei, Jiawei Han, B. Mortazavi-Asl, H. Pinto, Qiming Chen, U. Dayal, and Mei-Chun Hsu. «PrefixSpan,: mining sequential patterns efficiently by prefix-projected pattern growth». In: *Proceedings 17th International Conference on Data Engineering*. 2001, pp. 215–224. DOI: 10.1109/ICDE.2001.914830 (cit. on p. 13).

[4] R. Agrawal and R. Srikant. *Mining sequential patterns*. 1995 (cit. on p. 13).

[5] URL: https://www.kaggle.com/datasets/benhamner/sf-bay-area-bike-share (cit. on p. 23).

[6] URL: http://www.bayareabikeshare.com/ (cit. on p. 23).

[7] Luca Cagliero Luca Colomba and Paolo Garza. «Mining Spatiotemporally Invariant Patterns». In: *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '22. Seattle, Washington: Association for Computing Machinery, 2022. ISBN: 9781450395298 (cit. on p. 28).