POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Graphs of Objects: Innovating Data Modelling for Road Pavement Detection

Tesi di Laurea Magistrale in
Mathematical Engineering - Ingegneria Matematica

Author: **Marco Poggi**

Student ID: 10687966
Advisor: Prof. Simone Vantini
Co-advisors: Arianna Burzacchi
Academic Year: 2022-23

Abstract

The classification of road pavement is a fundamental step for the development of an efficient and resilient transportation system. However, this aspect is often overlooked, especially in developing countries.

As a consequence of this need, the goal of this work, in collaboration with the Safari Njema project at Politecnico di Milano, is the design of algorithms capable of classifying urban roads as either paved or unpaved based on publicly available information. This information has a dual nature: polygons identified by spatial coordinates and satellite images.

The first step is the creation of an innovative data structure capable of effectively representing this pair of sources of information. The concept of Graphs of Objects (GoOs) is introduced, where satellite images are represented as three-dimensional objects in the RGB space, and polygon boundaries are converted into undirected edges of a graph.

Subsequently, two algorithms of different nature are selected, a κ-Nearest Neighbors and a Graph Convolutional Network, and then adapted for the study of Graphs of Objects. These modified versions outperform their original counterparts, based respectively on objects and edges, demonstrating both the necessity of a GoO-based approach and the ability of the final algorithms to take into account all available information.

This project focuses specifically on the Greater Maputo area, Mozambique, but is designed to be extended to any other road network and potentially to Graphs of Objects of completely different nature.

Keywords: road pavement; classification; object-oriented; graph theory; κ-nearest neighbors; graph convolutional network

Abstract in lingua italiana

La classificazione del manto stradale è un passo fondamentale per lo sviluppo di un sistema di trasporti che sia efficiente e sicuro. Tuttavia, questo aspetto viene spesso tralasciato, specialmente in Paesi in via di sviluppo.

L'obiettivo di questo lavoro, in collaborazione col progetto Safari Njema del Politecnico di Milano, è quindi il disegno di algoritmi in grado di classificare le strade di un centro urbano in asfaltate e sterrate a partire da informazioni di pubblico dominio. Tali informazioni sono di duplice natura: poligoni identificati da coordinate spaziali e immagini satellitari.

Il primo passo è la creazione di una struttura dati innovativa, capace di rappresentare questa coppia di fonti di informazioni in maniera valida. Viene quindi presentato il concetto di Grafi di Oggetti (Graphs of Objects, GoOs), dove le immagini satellitari vengono rappresentate come oggetti tridimensionali nello spazio RGB e i confini tra poligoni sono convertiti in archi adirezionati di un grafo.

In seguito, vengono selezionati due algoritmi di diversa natura, un κ-Nearest Neighbors e una Graph Convolutional Network, i quali vengono poi adattati allo studio di Grafi di Oggetti. Queste versioni modificate performano meglio delle loro controparti originali, basate rispettivamente solo su oggetti e solo su archi, dimostrando sia la necessità di un approccio basato sui GoOs, sia la capacità degli algoritmi finali di tenere conto di tutte le informazioni disponibili.

Questo progetto si focalizza in particolare sulla provincia di Maputo, capitale del Mozambico, ma è pensato per essere esteso a qualunque altra rete stradale e, potenzialmente, anche a Grafi di Oggetti di natura completamente diversa.

Parole chiave: manto stradale; classificazione; object-oriented; teoria dei grafi; κ-nearest neighbors; graph convolutional network

Contents

Introduction

0.1. Context

In 2015, when the United Nations introduced the Sustainable Development Goals (SDGs, [1]), they emphasized the critical role of efficient infrastructure systems in cities worldwide. Two specific goals, Goal 9 ("Industry, Innovation, and Infrastructure") and Goal 11 ("Sustainable Cities and Communities"), center their attention on this aspect. One of their key objectives is to establish a secure, resilient, and sustainable public transportation system. While residents in developed nations should focus on adopting environmentally friendly solutions in their everyday lives, the challenges in developing regions, such as Sub-Saharan countries, are more deeply rooted.

To ensure the provision of a safe public transportation system for the population, the initial, often overlooked step is the collection of reliable data regarding the road infrastructure in major cities. It is not uncommon for roads to be paved without proper documentation, resulting in unreliable and incomplete data ([21]). This issue holds significant importance since data serves as the foundational element required to optimize any type of service. For this reason, the primary objective of this thesis was to design algorithms for the automatic pavement detection of the road network in Sub-Saharan cities, with a specific focus on the Greater Maputo area in Mozambique. This work develops within the Safari Njema project of Politecnico di Milano ([2]), which aims to optimize paratransit mobility in developing countries and redesign existing transportation for greater efficiency.

Automatic analysis of road pavement is not new in literature, but it is essential to note that most of the existing work focuses on issues such as crack detection ([33], [20]) or surface condition assessment ([36]). The problem we aimed to tackle was broader: classifying roads into two categories ("paved" and "unpaved"), in order to boost the efficiency of future optimization studies. A fundamental reference for this type of analysis is [10], to which this thesis serves as a spiritual sequel. The authors utilized Google Earth ([3]) satellite images and developed an advanced κ-Nearest Neighbors (κ-NN, [17]) algorithm based on those images. In the following pages, we will build upon and enhance those

Figure 0.1: Mismanaged public transport system (*Source*: Safari Njema)

results by incorporating other publicly available information, specifically the shape and location of each road segment, available via OpenStreetMap (OSM, [4]).

Due to the bipartite nature of the data, encompassing both images and the network structure, this thesis serves as a bridge between two highly prolific areas of Data Science: Object-Oriented Data Analysis (OODA, [14]) and Graph Theory ([7]). We will design a new type of data structure, called Graph of Objects (GoO), consisting of complex nodes, referred to as objects, connected with each other by undirected and unweighted edges.

This innovative approach opens up exciting possibilities across various domains, extending beyond our current application. Consider, for instance, environmental monitoring, where Graphs of Objects could facilitate the assessment of intricate ecosystems, capturing the interconnections between species and environmental factors. Similarly, in healthcare, this data structure could be used to connect medical images to the often complicated patient histories. The algorithms developed in this thesis serve as a first step toward harnessing the potential of GoOs in these and many other fields, promising novel data-driven solutions for complex challenges.

0.2. Content

The two algorithms that will be presented in this thesis are a κ-NN and a Graph Convolutional Network (GCN, [22]). Both represent an evolution of the work in [10], with the former being more directly connected to the original work, while the latter exploits important results to enhance the quality of a completely different approach. The aim is to extend the capabilities of these already established classification algorithms to our new bipartite type of data, thus showcasing the immense potential of GoOs.

These two algorithms originate from different domains, namely traditional Statistical Learning and Deep Learning, and will address the classification task in distinct ways. Our objective extends beyond the mere design of these algorithms; we also aim to compare and contrast their strengths and weaknesses, providing valuable insights into the two approaches.

The structure of the thesis is outlined as follows, with each chapter serving a distinct purpose and contributing to the overall narrative:

- **Chapter 1, Data Extraction and Preprocessing**: This chapter delves into the initial stages of our research, presenting a comprehensive examination of the available data regarding the city of Maputo, Mozambique. It provides insights into the data extraction and preprocessing processes for the GoO, forming the basis of our analysis.
- **Chapter 2, Theoretical Foundations**: This chapter provides a comprehensive review of theories related to Supervised, Unsupervised and Semi-Supervised Learning, emphasizing their relevance to our specific classification task. It sets the theoretical basis common to the subsequent chapters.
- **Chapter 3, κ-Nearest Neighbors**: In this chapter, we conduct a detailed exploration of the first algorithm, κ-NN. Starting from its inception, we trace its development and implementation, ultimately arriving at the results achieved when applied to the GoO.
- **Chapter 4, Graph Convolutional Network**: Turning our attention to the second algorithm, the GCN, this chapter explores its conceptualization and its practical application to the GoO. We delve into the theoretical underpinnings and practical aspects of this Deep Learning-based algorithm, unveiling its potential in our context.
- **Chapter 5, Compared Approaches**: This chapter critically compares and contrasts the outcomes of both algorithms, shedding light on their strengths and weaknesses. It examines key aspects such as performance, generalizability, computational time, and visual result.

In the end, a brief concluding chapter summarises the findings of the entire thesis work and an appendix presents the code related to the computations performed, so that the reader may be able to reproduce our results and apply the algorithms to other GoOs.

1 | Data Extraction and Preprocessing

1.1. Data Sources

Following the approach presented in [10], the analysis relied on two primary data sources: the OpenStreetMap (OSM) road network and Google Earth high-resolution satellite images. In [10], the OSM road network underwent segmentation, resulting in 53240 distinct segments. Each segment represented a road stretch spanning a length between 50 m and 550 m. Paths shorter than this threshold were omitted from consideration, while longer ones were divided into sub-sections. Throughout our research, we consistently employed this segmentation to facilitate direct comparisons with the original work.

OSM provided various attributes for each street segment, including universally available ones such as road length (`length`) and street type (`osm_type`). However, for the purposes of our analysis, we exclusively focused on Graphs of Objects (GoOs). Consequently, most of this available information was disregarded. The essential attributes that we retained were the geometry of the network (`geometry`) and the road surface type (`osm_surf`).

The `geometry` attribute contained information about the shape and location of each road segment in the network and was designed to be accessed using the `sf` R library ([30]), specifically tailored for simple features. Roads were interpreted as polygons identified by sequences of points forming a closed, non-self-intersecting ring. Each of these points was associated with two-dimensional coordinates, enabling the study of the relative location of each segment in relation to all others. The coordinate reference system used was the World Geodetic System 1984 (WGS84, [26]), typically three-dimensional but projected into two dimensions, X and Y, for this particular analysis. `geometry` will serve as the primary data source for creating the final graph.

Meanwhile, `osm_surf` indicated whether a road was paved or unpaved. More specific surface tags, such as "asphalt", were aggregated into the two primary classes. Notably, fewer than 5% of the roads exhibited a well-defined pavement composition. To be precise, 732 streets were classified as paved, while 1826 were designated as unpaved. `osm_surf` will serve as the response variable for both our algorithms.

In Figure 1.1 it is possible to see the original OSM dataset. The scarce amount of colored roads represents the labeled data, while everything else needed to be classified. It can be noticed that some specific areas, like the coast in the eastern part of the map, present an high concentration of labeled data points.This will be crucial for the first part of our analysis, and we will revisit this aspect shortly.

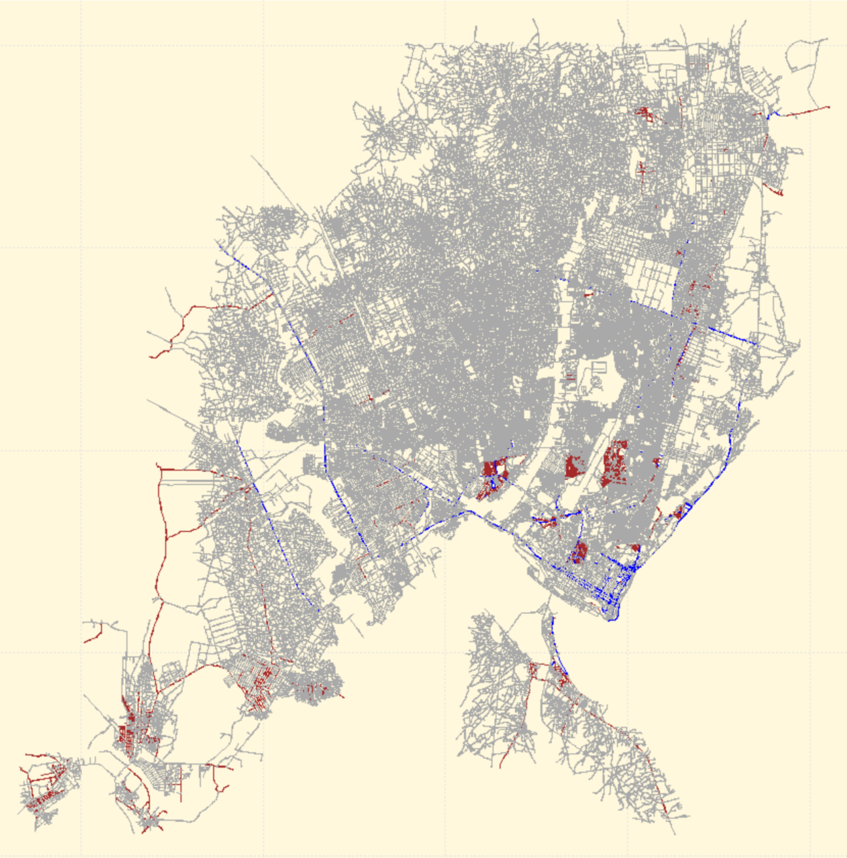

Figure 1.1: Original `osm_surf` labeling

🟦 Paved 🟥 Unpaved ⬛ Unknown

Regarding the satellite images, these were acquired from Google Earth and covered the entire Greater Maputo area, including not only the city of Maputo but also the neighboring districts of Marracuene, Matola, and Boane. This extensive coverage encompassed a total area of 1568 km^2. To ensure precision in the subsequent analysis, the satellite data underwent a clipping process, executed using QGIS software ([32]). Consequently, a grand total of 53240 distinct raster images were generated, with each image aligning precisely to a specific segment of the OSM road network.

1.2. Objects

To conversion from satellite images to objects, was done according to what was presented in [10]. We will not delve deeply in the entire pipeline producing such objects since it was not the focus of our work and we advise the consultation of the original work. In essence, every image, after being clipped to fit its related polygon in the OSM dataset, was also filtered to remove unwanted areas (generally including vegetation or vehicles). This filtering process consisted of the removal of the darkest pixels and the following application of the DBSCAN ([16]) algorithm, which was proven to further improve results. Of the remaining pixels, 150 were randomly sampled and interpreted as three-dimensional points in the RGB space, identifying an object. RGB stands for Red, Green and Blue and it is a model used in digital imaging and computer graphics to represent colors. In this model, colors are created by combining different intensities of red, green and blue light. Each of these three primary colors can have a range of values, typically from 0 to 255.

From now on, the set of all objects will be called Ω. Because of the paramount importance of labeled data in every Supervised and Semi-Supervised Learning ([39]) algorithm, we also gave a name to the subset of labeled objects: $\Omega_{\mathcal{L}}$. Furthermore, we defined $\Omega_{\mathcal{P}}$ and $\Omega_{\mathcal{U}}$, subsets of known paved and unpaved roads, such as

$$\Omega_{\mathcal{P}} \cup \Omega_{\mathcal{U}} = \Omega_{\mathcal{L}} \subset \Omega \tag{1.1}$$

The cardinalities of these sets will be respectively referred to as N, $N_{\mathcal{L}}$, $N_{\mathcal{P}}$ and $N_{\mathcal{U}}$. By definition,

$$N_{\mathcal{P}} + N_{\mathcal{U}} = N_{\mathcal{L}} < N \tag{1.2}$$

In Figure 1.2, we illustrate the entire image processing pipeline for a paved and an unpaved road. In this particular case, the difference between pavement evidently reflects itself in the shape of the final objects, hinting that this could be a successful strategy. In order to mathematically demonstrate the effectiveness of this approach for discriminating objects, it was necessary to define a measure of dissimilarity between two clouds in the RGB space. After several trials, it was discovered in [10] that the best-performing dissimilarity measure was the energy distance ([37]), defined as:

$$d_e(\omega_i, \omega_j) = \frac{n_i n_j}{n_i + n_j} \left[\frac{2}{n_i n_j} \sum_{a \in \omega_i, b \in \omega_j} d(a, b) - \frac{1}{n_i^2} \sum_{a, a' \in \omega_i} d(a, a') - \frac{1}{n_j^2} \sum_{b, b' \in \omega_j} d(b, b') \right] \tag{1.3}$$

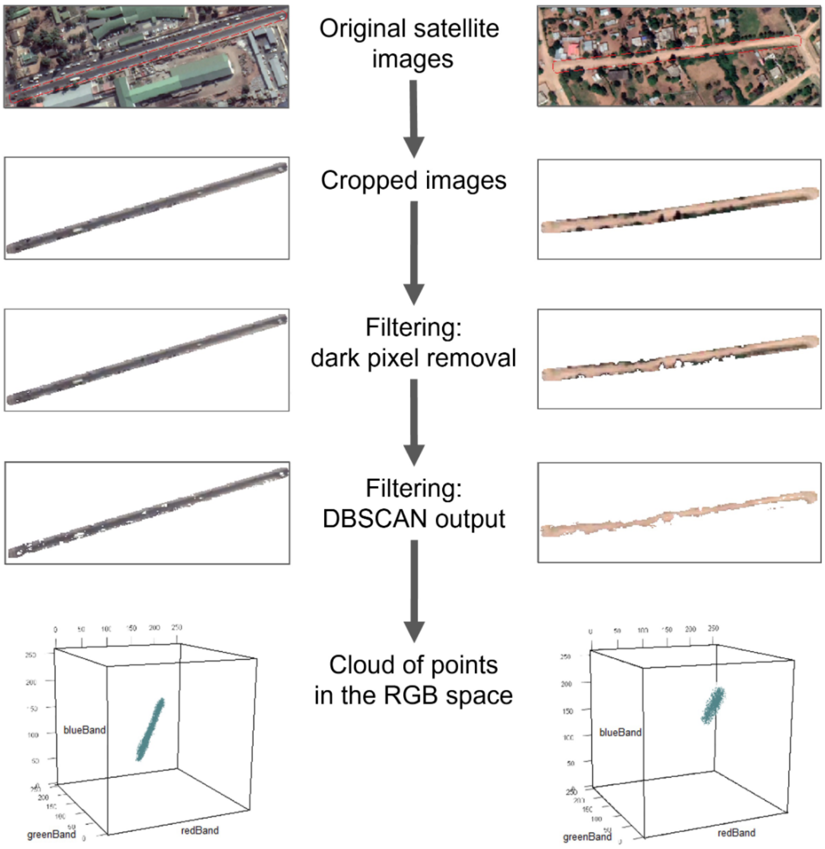
Original satellite images
Cropped images
Filtering: dark pixel removal
Filtering: DBSCAN output
Cloud of points in the RGB space
blueBand
greenBand
redBand

Figure 1.2: Image processing pipeline (*Source*: [10])

where $\omega_i, \omega_j \in \Omega$, and n_i, n_j are the numbers of pixels in each object (in our case, 150). d is the so-called ground distance, and it was chosen to be the Euclidean distance in the RGB space:

$$d(a,b) = \sqrt{(R_a - R_b)^2 + (G_a - G_b)^2 + (B_a - B_b)^2} \tag{1.4}$$

where a and b are two pixels, and the variables on the right-hand side represent their values of red, green, and blue.

In [10], the energy distance (1.3) was the metric used for the final κ-NN algorithm. In this work, we will explore ways to incorporate this information about the objects into more complex dissimilarity measures, taking into account the edges of the GoO as well.

What's important to understand from the preprocessing presented in this paragraph is that it was not necessary to include information on every single pixel of every road segment in our analysis. What needed to be retained was the dissimilarity measure between elements of Ω. Specifically, we computed:

$$E_{ij} = d_e(\omega_i, \omega_j) \tag{1.5}$$

an $N \times N_{\mathcal{L}}$ matrix, where $\omega_i \in \Omega$ and $\omega_j \in \Omega_{\mathcal{L}}$.

1.3. Edges between Objects

Until now, we only considered the images associated with each object $\omega_i \in \Omega$, but the bipartite nature of our data must not be forgotten. As previously stated, each road was also associated with a polygon made of points, each point associated with two-dimensional WGS84 coordinates. To make full use of this type of information, we identified all intersections between different polygons and converted them into the edges of our final GoO. The idea was that the presence of a border between two roads was likely related to an equal type of pavement. We called the set of all edges H and each $\eta_{ij} \in \mathrm{H}$ was defined as

$$\eta_{ij} = \mathbb{1}\{\omega_i \cap \omega_j \neq \emptyset\} \tag{1.6}$$

Clearly, $\forall \omega_i \in \Omega$

$$\eta_{ii} = 1 \tag{1.7}$$

In practice, the intersection between polygons was analyzed using the `st_intersects` R command, once again from the `sf` R library. During this process, it became evident that almost the entire dataset could be represented in a single graph, but a limited number of roads appeared disconnected from it. In Figure 1.3, it is possible to see a specific area where this inaccuracy is evident.

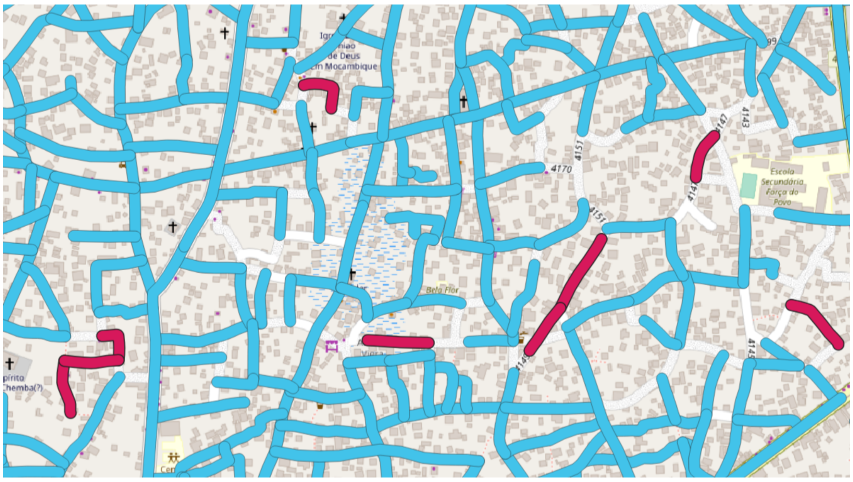

Figure 1.3: Some unconnected roads

🟦 Main graph 🟥 Unconnected roads

It seemed quite clear that some real roads in Maputo were never added to this dataset, resulting in this unusual behavior of having some isolated roads. Since obviously the original idea of this network was to be connected and only 0.54% of the roads were not included in the main sub-graph, we thought that the most straightforward solution was to simply delete this anomalies from the dataset and work with the remaining roads. As a consequence of this, from now on $N = 52953$ and $N_{\mathcal{L}} = 2535$.

Being the edges we computed undirected and unweighted, they could be stored in an adjacency matrix:

$$A_{ij} = \eta_{ij} \tag{1.8}$$

We will see that for the correct functioning of our algorithms, it will be important to store the entire $N \times N$ adjacency matrix A. Since this computation is significantly less time consuming than the computation of the energy matrix E, this was not an issue.

1.4. Graphs of Objects

At this point, satellite images had been utilized for creating the objects Ω, while the `geometry` attribute had been used to design the edges H. The next natural step was to combine these elements to form our GoO:

$$\Gamma(\Omega, \mathrm{H}) \tag{1.9}$$

In Figure 1.4, it is possible to observe a representation of six roads of Γ along with their respective objects and edges. Each object is a 3D cloud of pixels in the RGB cube, while the edges are simple undirected, unweighted connections formed by intersections between polygons. To enhance the clarity of the image, edges of the type η_{ii} have not been visually represented, although they are indeed present, since by definition a node is considered adjacent to itself.

It's worth emphasizing that for our classification purposes, we won't primarily rely on $\Gamma(\Omega, \mathrm{H})$ itself, but on the matrices E and A computed in the previous sections and respectively representing information hidden in Ω and H. However, we stress the GoO structure to highlight that we won't be using any synthetic index related to individual road segments, but rather dissimilarity and similarity measures between complex multi-dimensional objects.

During our study, we also considered representing data with a more traditional graph structure: a simple node would have been assigned to each road segment and information included in E would have been used to differently weigh the edges. However, we decided

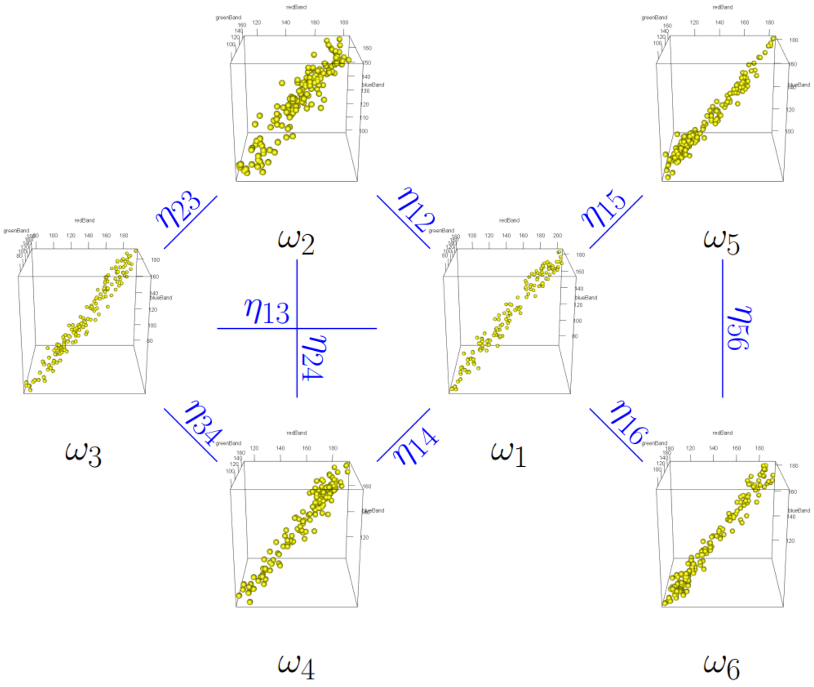

Figure 1.4: Six roads of Γ(Ω,H)

that having the nodes represent dissimilarity between images, while the edges only encapsulate information from the OSM network, was more coherent with our dual data sources and therefore made more sense.

Ω and its subsets $\Omega_{\mathcal{L}}$, $\Omega_{\mathcal{P}}$ and $\Omega_{\mathcal{U}}$ are characterized by the cardinalities in 1.1.

Variable	Value
N	52953
$N_{\mathcal{L}}$	2535
$N_{\mathcal{P}}$	729
$N_{\mathcal{U}}$	1806

Table 1.1: Cardinalities in $\Gamma(\Omega, \mathrm{H})$

Due to the substantial amount of available data and the complexity of handling a full $N \times N$ adjacency matrix A, it became necessary, at least for the initial stages of our analysis, to create a limited dataset covering only a specific area of the city. The region selected corresponded to a coastal area of the center of Maputo, it will be referenced as

$$\Gamma'(\Omega', \mathrm{H}') \tag{1.10}$$

and it can be seen in Figure 1.5.

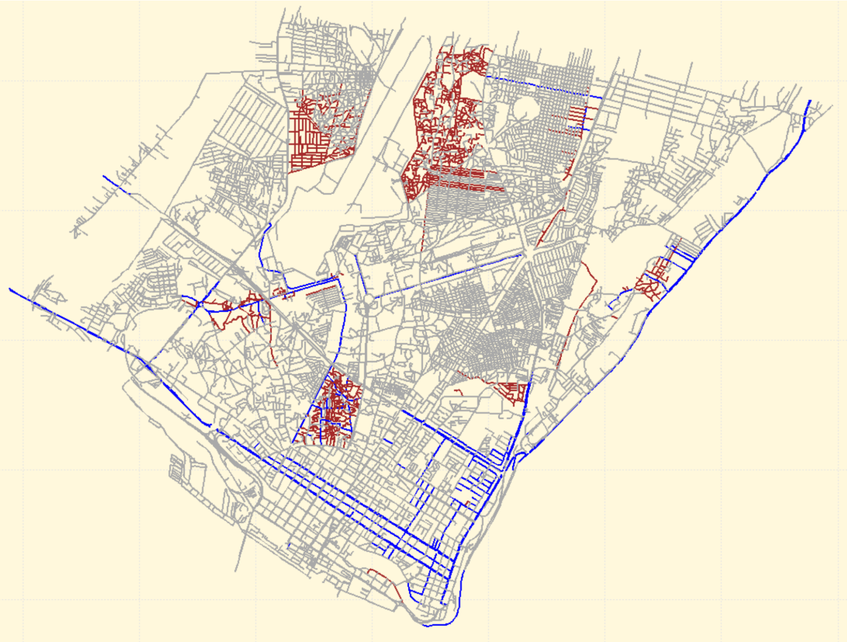

Figure 1.5: `osm_surf` labeling for the limited dataset Γ'

🟦 Paved 🟥 Unpaved

Ω' and its subsets $\Omega'_{\mathcal{L}}$, $\Omega'_{\mathcal{P}}$ and $\Omega'_{\mathcal{U}}$ are characterized by the cardinalities in 1.2.

Variable	Value
N'	6065
$N'_{\mathcal{L}}$	1163
$N'_{\mathcal{P}}$	336
$N'_{\mathcal{U}}$	827

Table 1.2: Cardinalities in $\Gamma'(\Omega', \mathrm{H}')$

This area was selected for a dichotomy of reasons. Firstly, like we already mentioned when analysing Figure 1.1, it offered a substantial amount of labeled data (19.17% of the entire GoO, compared to the 4.80% of Γ), enabling effective training without being hindered by an excessively large adjacency matrix. Secondly, the proportion $\frac{N'_{\mathcal{P}}}{N'_{\mathcal{U}}} = 0.406$ was nearly identical to the original $\frac{N_{\mathcal{P}}}{N_{\mathcal{U}}} = 0.401$, indicating an almost perfect representation of the entire dataset. Our algorithms will initially be tested using this limited amount of information, and only in the final phase their generalizability to the Greater Maputo area will be assessed.

2 | Theoretical Foundations

2.1. Learning Paradigms

The three fundamental learning paradigms in Statistics and Machine Learning are Supervised, Unsupervised, and Semi-Supervised Learning ([8], [25]). As these paradigms play a crucial role in our subsequent analysis, it is essential to provide a theoretical overview, emphasizing their relevance to our work.

2.1.1. Supervised Learning

Supervised Learning stands as the most widely embraced and well-established paradigm among the three. This research centers on deciphering data generated by an unknown function f that maps input X, commonly known as attributes or features, to a corresponding output $\underline{y}$, referred to as response or target:

$$f(X) = \underline{y} \tag{2.1}$$

The primary objective is to approximate the unknown function as closely as possible:

$$\underline{\hat{y}} = \hat{f}(X) \simeq f(X) = \underline{y} \tag{2.2}$$

To achieve this, the model is trained on a comprehensive dataset $\mathcal{D} = \{(X_i, y_i)\}_{i=1}^{N}$, consisting of input-output pairs. Each element in this dataset serves as a foundational block for the model's training process, facilitating the extraction of patterns and correlations. Through a systematic analysis of this data, the model iteratively refines its parameters, adjusting its internal mechanisms to enhance predictive capabilities. This iterative learning process aims to minimize the disparity between predicted output $\underline{\hat{y}}$ and actual output $\underline{y}$, resulting in a model that generalizes well to unseen data.

Generally, when the response $\underline{y}$ is continuous, the problem is termed *regression*; conversely, when it is discrete, it is referred to as *classification*. The function f that needs to be approximated in a classification task is called a decision boundary or decision surface. In Figure 2.1 an example of classification with a linear decision boundary is presented.

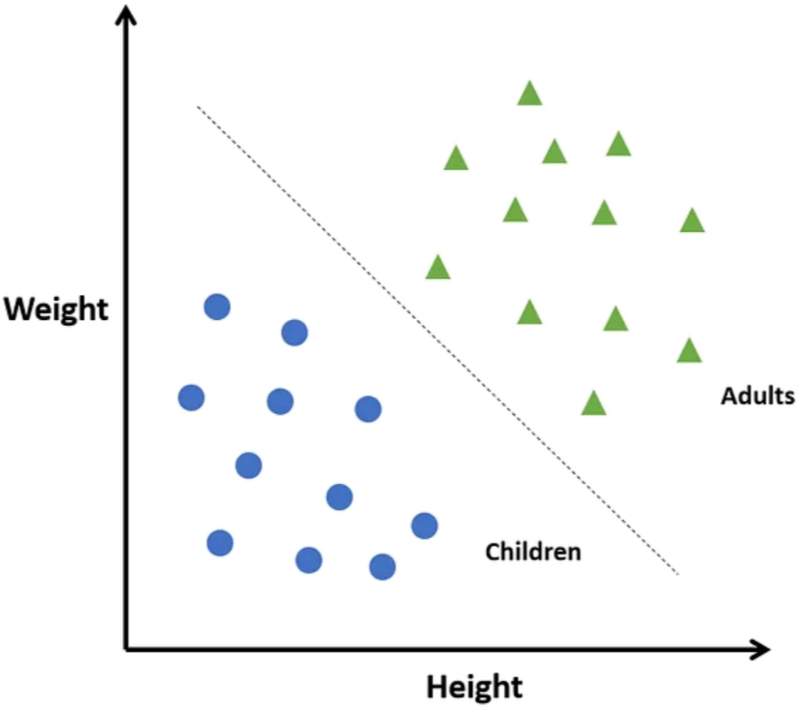

Figure 2.1: Classification on a toy problem (*Source*: Analytics Vidhia)

One prominent example of a classification problem to bear in mind while navigating this thesis is outlined in [10]. In that particular context, input was represented by the set of objects Ω in the three-dimensional RGB cube. This departure from classical Data Analysis, where features are typically structured in an $N \times F$ table (with F denoting the number of features), yielded promising results and unveiled novel avenues for Object-Oriented Data Analysis. Concerning the corresponding response, y_i, it denoted the membership class of the i-th element ω_i within Ω, aligning more closely with the conventional concept of a target variable.

The reader can now ascertain that the case presented in this thesis, involving the classification of roads in Maputo based on a GoO-structured dataset, serves as a direct continuation of the work in [10]. While one might instinctively categorize it within the domain of Supervised Learning, the inclusion of edges η_{ij} connecting $\omega_i \in \Omega_{\mathcal{L}}$ to $\omega_j \in \Omega \setminus \Omega_{\mathcal{L}}$ transforms it into an exemplary instance of Semi-Supervised Learning. This aspect will be scrutinized more comprehensively in the ensuing sections.

2.1.2. Unsupervised Learning

While Supervised Learning deals with labeled datasets and aims to predict a target variable, Unsupervised Learning operates on unlabeled data with the primary goal of uncovering hidden patterns and structures within the dataset. The absence of an explicit output $\underline{y}$ for the given input variables X introduces a distinctive challenge and opportunity. Without predefined labels guiding the model, the algorithm must autonomously discern inherent structures within the data.

One key task of Unsupervised Learning is *clustering*, also known as *unsupervised classification*. It involves grouping similar data points together, forming clusters or subgroups based on shared characteristics. Figure 2.2 presents an example of clustering performed over the same toy problem used for Figure 2.1.

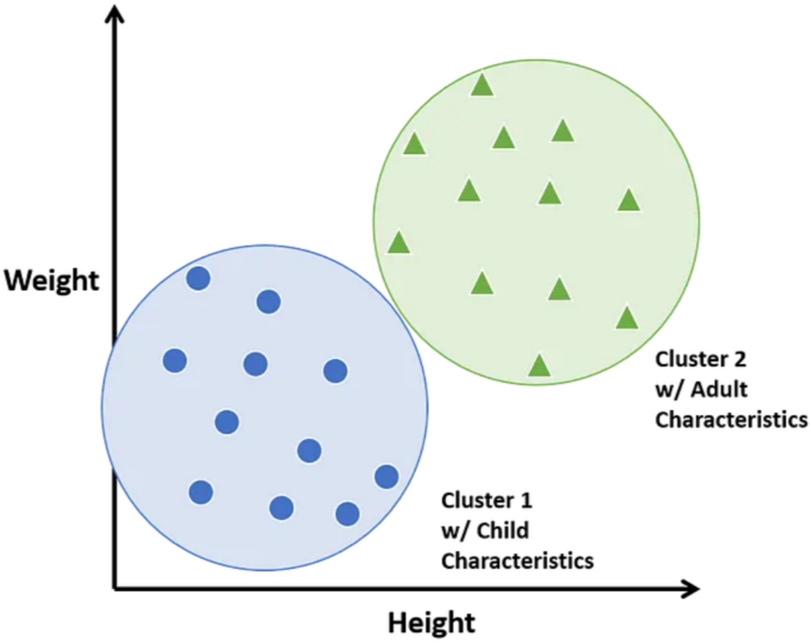

Figure 2.2: Clustering on a toy problem (*Source*: Analytics Vidhia)

While a wide variety of clustering methods and algorithms exist and are widely used, explaining all of them goes beyond the scope of our work. We will venture into the realm of *community detection* ([28]), a process extremely akin to clustering specifically tailored for graphs and networks. Such a process is capable of finding sub-graphs based solely on the structure of the adjacency matrix and edges. Belonging to the same sub-graph will adjust the similarity between two elements of Ω and prove fundamental for improving the performance of the κ-NN algorithm. However, a more in-depth exploration of the concept of *community detection* will be presented in Chapter 3.

2.1.3. Semi-Supervised Learning

A pioneering paper that explores the potential of a semi-supervised approach to learning is [35]. Intriguingly, the term "Semi-Supervised Learning" is conspicuously absent from the paper. Instead, the author opts for the phrase "Supervised Learning aided by additional unlabeled data". This encapsulates the essence of this learning paradigm perfectly. While the goal of a semi-supervised problem aligns with that of a supervised one, the two approaches diverge in the fact that the semi-supervised learner is supplied with both labeled and unlabeled data. This scenario is common in situations where unlabeled data is readily available, but obtaining labels proves costly. The optimism in this paradigm lies in the belief that there exists information, not too expensive to extract, capable of achieving superior performance compared to the one obtained in a supervised setting.

Graph Theory, especially *node classification on graphs*, is a domain where Semi-Supervised Learning is frequently employed ([39]). In this context, it's not unusual to be equipped with an entire graph structure but only a limited number of known labels. Relying solely on the characteristics of the labeled nodes is not practical; instead, it is assumed that labeling can be propagated in a specific manner throughout the graph. Analyzing the complete structure of the graph becomes paramount and extensive work has been undertaken in the literature to devise optimal methods for modeling this propagation. The initial methods explored were broadly categorized into two groups: those incorporating some form of explicit graph Laplacian regularization ([39], [5]) and those based on graph embeddings ([24], [38]). However, in recent years, the emergence of Artificial Neural Networks (ANNs) has significantly broadened the horizons. An illustrative example is provided by [22], the groundbreaking work that introduced Graph Convolutional Networks and will serve as an essential element for constructing our second algorithm in Chapter 4.

By now, the reader likely perceives how the inclusion of edges η_{ij}, connecting $\omega_i \in \Omega_{\mathcal{L}}$ to $\omega_j \in \Omega \setminus \Omega_{\mathcal{L}}$, transforms our analysis into a Semi-Supervised version of the work in [10]. This enables the dissemination of labeling information across the entire GoO.

2.2. Learning Stages

In Chapter 1, we discussed the design and extraction of the dataset, $\mathcal{D} = (\Gamma, \underline{y})$, where Γ was the GoO and $\underline{y}$ was the response represented by `osm_surf`, and the selection of the algorithms, namely κ-NN and GCN. The lingering question is how an algorithm $\mathcal{A}$ can effectively learn from data. To address this, we present a schematic overview of the stages in a typical learning process, providing insights into how each step relates to the specific case at hand. [25] was of great inspiration for this paragraph.

1. **Data Partitioning**: Labeled data is divided into three subsets called *training*, *validation* and *test* set.

   - **Training**, as the name suggests, is the set actually used to train the learning algorithm $\mathcal{A}$ and it is the only set of input-output pairs that can be used from the beginning to predict f.

     The algorithm $\mathcal{A}(\Theta)$ depends on a set of so-called *hyperparameters* Θ. These free parameters are distinct from the one learned by the algorithm because they are given as inputs by the user.

   - **Validation** is used to tune these hyperparameters Θ. This tuning is a form of *model selection*, because it allows to find the optimal version of the algorithm, $\mathcal{A}(\Theta_0)$.

     It is crucial that this model selection is done on a set different from the training one, to avoid an issue called *overfitting.* This behavior occurs when the model gives accurate predictions for training data but not for new unseen data. It is caused by the use of the same set of data for training and assessment and it is generally characterized by the choice of a model that is too complex and depends on too much parameters.

   - **Test** is used to evaluate the performance on unseen data. This obviously can't be done on the training set for the reasons just mentioned, but it can't be done on the validation set either, in order not to overfit the selection of Θ_0.

     For the model, the test set is basically equivalent to any unlabeled data. The only difference is that it can be used to compute accuracy, precision and many other metrics useful to express the correctness of the provided results.

   Typically, the training set is considerably larger than the other two sets, yet the precise percentage hinges on the algorithm type and the available data size. The volume of data in the validation set is primarily determined by the number of *hyperparameters.* As for the test set, it encompasses the remaining data, and its size must be substantial enough to provide an accurate assessment of performance on unseen data. If the original data presents some unbalance between classes (e.g. a binary problem with 80% of instances belonging to class 0 and 20% belonging to class 1) such unbalance needs to be mirrored in the three sets. In this way, each of them will be representative of the entire population, avoiding unwanted bias.

2. **Feature Selection**: Moving forward, our attention turns to selecting relevant features associated with the labeled data points. This stage underscores the pivotal role of features in guiding the learning algorithm. The user's prior knowledge comes into play, influencing the choice of features. This selection process forms the base for effective algorithmic learning.

   The realm of feature selection is extensive and a perpetual subject of study. However, as emphasized in our thesis, features in our context significantly differ from the conventional notion. In our case, the only selection required was deciding to summarize the set Ω with the energy distance matrix E.

   Had this not been a thesis focusing on Object-Oriented Data Analysis, an exhaustive feature selection process would have been essential. This would have involved deciding which synthetic indices to retain to describe each satellite image, leading to a more traditional $N \times F$ feature table X. For instance, metrics like the mean and variance of red, green and blue RGB values could have proven effective for classifying a road as "paved" or "unpaved".

3. **Algorithm Training**: This constitutes the heart of the learning process. As previously mentioned, the chosen algorithm undergoes training on the training set, considering all possible combinations of hyperparameters Θ. For each parameter set, the algorithm selects a distinct hypothesis from the *hypothesis set*, which encompasses all possible $\hat{f}$. Subsequently, performance evaluation transpires on the validation set to pinpoint the optimal Θ_0. The assessment of performance can be executed through various metrics. It may involve minimizing a *loss function* or an *error*, or maximizing statistics such as accuracy and precision.

   For regression problems, a widely used loss function is the *Sum of Squared Errors (SSE)*:

$$\mathcal{L}_{SSE} = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.3}$$

   Here, N represents the dimension of the validation set, and the loss is intuitively smaller when the chosen hypothesis $\hat{f}$ aligns well with the real function f. In contrast, for classification problems, a common choice for the loss function is the *Cross-Entropy (CE)*. It's essential to note that many classification algorithms (e.g., both κ-NN and GCN) don't directly output a discrete class but a continuous probability Z_i for each class. The class with the maximum probability is then chosen as the prediction $\hat{y}_i$, defined as:

$$\hat{y}_i = \max_{c \in C} Z_{i,c} \tag{2.4}$$

where C represents the set of possible classes. In this context, the cross-entropy loss is defined as:

$$\mathcal{L}_{CE} = -\sum_{i=1}^{N} \sum_{c \in C} y_i \log Z_{i,c} \tag{2.5}$$

which decreases as the certainty towards corrects predictions increases.

Conversely, if the user opts to maximize a statistic, such as accuracy, the process solely considers the coherence between the predicted class $\hat{y}_i$ and the true class y_i.

In this work, multiple optimization criteria will be employed at different stages of the two algorithms to ensure the most thorough assessment. A more context-specific discussion of optimization will be presented in the following paragraphs.

4. **Prediction and Evaluation**: This is the final phase of the process and it is the one applied on the test set. As mentioned, the test set represents unseen data and it is kept to compute the quality of $\mathcal{A}(\Theta_0)$. The same assessment criteria presented in the previous stage are applied to the test set ensuring a robust understanding of the algorithm's capabilities.

Figure 2.3 represents the learning stages in a schematic and more intuitive way.

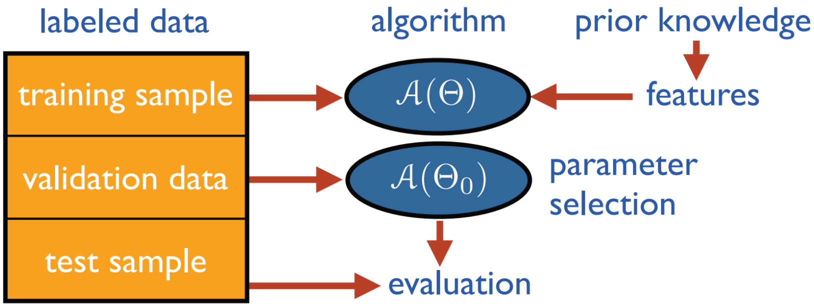

Figure 2.3: Illustration of the typical stages of a learning process (*Source*: [25])

2.3. Cross-Validation

We have previously emphasized why hyperparameters should not be chosen based on their performance on the training set. However, the availability of data may not always be sufficient to create adequately sized training, validation, and test sets. Moreover, as mentioned earlier, the validation set itself can be susceptible to overfitting, rendering the optimal set of parameters Θ_0 less reliable. In light of these considerations, the concept of *S-fold cross-validation* comes into play. The premise is straightforward: the available data is initially split into a training set and a test set. Subsequently, the training data is partitioned into S groups, usually of equal size. The model is then trained using $S-1$ groups, and the results are evaluated on the remaining group. This process is repeated for all S possible choices for the held-out group, and the performance scores from these runs are averaged. This averaging helps significantly reduce variance. In cases of extremely limited data, the choice is often $S = N$, where N is the total number of data samples, leading to *leave-one-out cross-validation*.

The technique is illustrated schematically for $S = 3$ in Figure 2.4.

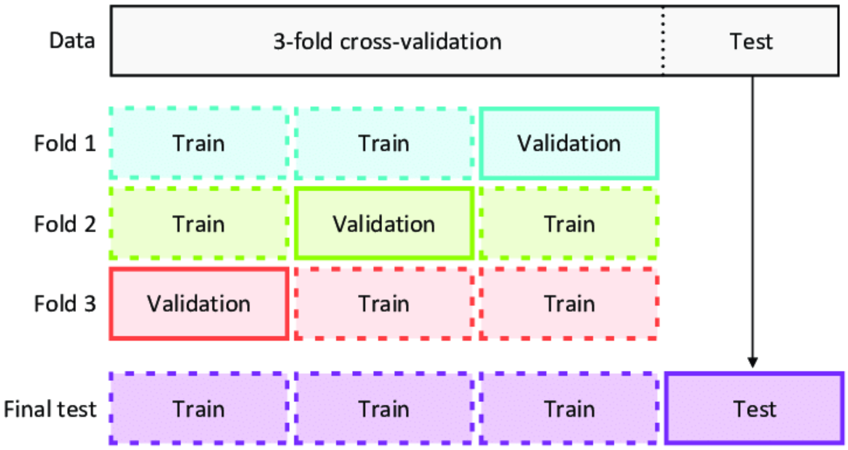

Figure 2.4: Dataset split for 3-fold cross-validation (*Source*: [15])

One drawback of cross-validation is the increasing number of training runs with the number of folds. If an algorithm is computationally expensive and there is a large number of possible hyperparameter combinations, performing cross-validation might become unaffordable. The user must find a suitable trade-off between accuracy and computational time by selecting an appropriate value for the number of folds.

As we will demonstrate, we will employ cross-validation in the initial part of our analysis when training the models on $\Gamma'(\Omega', \mathrm{H}')$. This choice is not only driven by the limited number of samples but also aims to minimize variance in the results. Subsequently, we will use the optimized hyperparameters Θ_0 in the much larger $\Gamma(\Omega, \mathrm{H})$, where performing cross-validation becomes prohibitive due to significantly slower computational times resulting from the vastly larger adjacency matrix.

2.4. Assessment Criterion

We mentioned the fact that there exist multiple ways of assessing the quality of a model. The maximization of the accuracy or other metrics was one of those, but also the minimization of a loss or an error was a viable option. Throughout this thesis, we opted for the last option and decided to define an error capable of summarizing the quality of our performance. Once again, [10] was of great inspiration for the definition of such error.

Both our algorithms will produce as output an $N \times 2$ matrix Z, where each row Z_i contains the probabilities for data point i to belong to class 0 ("paved") and class 1 ("unpaved"). By definition we will have:

$$\begin{cases} Z_{i,1}, Z_{i,2} \in [0,1] \\ Z_{i,1} + Z_{i,2} = 1 \end{cases} \tag{2.6}$$

At this point we will assign a predicted class to each data point i according to the following rules:

$$\hat{y}_i = \begin{cases} 0 & \text{if } Z_{i,1} > \theta_2 \\ 1 & \text{if } Z_{i,1} \leq \theta_1 \\ 2 & \text{otherwise} \end{cases} \tag{2.7}$$

where 2 corresponds to a third class referred to as "uncertain". We will investigate further the role and the value of the thresholds θ_1 and θ_2 in the next chapters. For now, it is only meaningful to understand how the continuous values of Z will be converted into classes.

From this classification, it will be possible to build the *confusion matrix* presented in Table 2.1, indicating the quantity and entity of the mistakes made by the algorithms. Coherently with the work in [10], the rows represent the true classes, while the columns represent the predicted ones.

		Predicted		
		0	1	2
True	0	N_{00}	N_{01}	N_{02}
	1	N_{10}	N_{11}	N_{12}

Table 2.1: Confusion matrix in the general case

Keeping this in mind, we decided to define an error based on the various types of misclassification, giving to each one of those a different *misclassification cost*. The values chosen are in Table 2.2. To assign meaningful values to these costs, certain considerations rooted in the real-life scenario under investigation were imperative. The misclassification costs

Misclassification	Cost
$y_i = 0, \hat{y}_i = 1$	2
$y_i = 0, \hat{y}_i = 2$	1
$y_i = 1, \hat{y}_i = 0$	2.5
$y_i = 1, \hat{y}_i = 2$	1

Table 2.2: Misclassification costs

had to be asymmetrical, as intuitively, categorizing a road as "uncertain" was deemed less problematic than completely misclassifying it. Moreover, a paved road misclassified as unpaved could potentially cause less damage than an unpaved road misclassified as paved. This is due to the fact that the entire public transport system, which we aimed to optimize, would be adversely affected if an impassable road were mistakenly added to its routes. All this reasoning lead to the definition of this error based on the confusion matrix:

$$\text{err} = 2N_{01} + N_{02} + 2.5N_{10} + N_{12} \tag{2.8}$$

An error defined like this posed two main concerns. The first one was that it strongly depended on the total number of roads being tested. Of course this was not an option, because we wanted to start with a limited dataset $\Gamma'(\Omega', \mathrm{H}')$ and then extend our results to the entire Greater Maputo area encapsulated in $\Gamma(\Omega, \mathrm{H})$. An error defined like this was obviously going to be much bigger on Γ and it would have been really difficult to compare the results. The second issue was that it was not possible to get an idea of how well the models were performing simply looking at (2.8). We needed to set a benchmark and check the ratio between our error and the one obtained in the worst case scenario.

To solve both these problems, we relied on the concept of *dummy classifiers*. These classifiers consistently predict the same label, irrespective of the data point, and three variations were possible due to the three potential values of the response. The confusion matrices for these dummy classifiers are displayed in Tables 2.3, 2.4, and 2.5.

		Predicted		
		0	1	2
True	0	$N_{\mathcal{P}}$	0	0
	1	$N_{\mathcal{U}}$	0	0

Table 2.3: Confusion matrix using the first dummy classifier

The error associated with the dummy scenario is the minimum achieved by any of these

		Predicted		
		0	1	2
True	0	0	$N_{\mathcal{P}}$	0
	1	0	$N_{\mathcal{U}}$	0

Table 2.4: Confusion matrix using the second dummy classifier

		Predicted		
		0	1	2
True	0	0	0	$N_{\mathcal{P}}$
	1	0	0	$N_{\mathcal{U}}$

Table 2.5: Confusion matrix using the third dummy classifier

three classifiers:

$$\begin{aligned} \text{err}_{\text{dummy}} &= \min\{\text{err}_{\text{dummy0}}, \text{err}_{\text{dummy1}}, \text{err}_{\text{dummy2}}\} && (2.9) \\ &= \min\{2.5N_{\mathcal{U}}, 2N_{\mathcal{P}}, N\} && (2.10) \end{aligned}$$

This allowed us to normalize our error, providing a clearer interpretation. The normalized error, which the algorithms aim to minimize, is referred to as the *pavement error* and is given by the equation:

$$\mathcal{E} = \frac{\text{err}}{\text{err}_{\text{dummy}}} \cdot 100\% \tag{2.11}$$

2.5. Input and Output of the Algorithms

We conclude this chapter with a brief recap of what the κ-NN and GCN will get as input and what the output will be. Everything was structured in order for the two algorithms to have the same starting point and the same target, in order to be able to compare and contrast their results in a second moment. For simplicity, here we present the names of input and output data without the apostrophe, but, as previously stated, everything will be first tried on Γ' and then applied to Γ.

An in-depth guide to the code used will be presented in Appendix A.

Input

- A: $N \times N$ matrix containing the edges H;
- E: $N \times N_{\mathcal{L}}$ matrix encapsulating the energy distance between pair of objects in $\Omega \times \Omega_{\mathcal{L}}$;

- $\underline{y}$: vector of N elements containing the true labels and defined as

$$y_i = \begin{cases} 0 & \text{if } \omega_i \text{ is paved} \\ 1 & \text{if } \omega_i \text{ is unpaved} \\ 2 & \text{if } \omega_i \text{ is unknown} \end{cases} \tag{2.12}$$

- Train - Test split: a random assignment of labeled data to the training set (80%) and the test set (20%). In both sets the proportion $\frac{N_{\mathcal{P}}}{N_{\mathcal{U}}}$ will be the same as in the entire $\Omega_{\mathcal{L}}$. Every version of each algorithm will be performed on different splits for consistency.

Output

- $\underline{\hat{y}}$: vector of N elements containing the predicted labels and defined as

$$\hat{y}_i = \begin{cases} 0 & \text{if } \omega_i \text{ is predicted as paved} \\ 1 & \text{if } \omega_i \text{ is predicted as unpaved} \\ 2 & \text{if } \omega_i \text{ is predicted as uncertain} \end{cases} \tag{2.13}$$

  It is worth mentioning that this value will be computed for each data point in Ω, but only the elements assigned to the test set will be used to compute the final error and assess the quality of the model;

- C: confusion matrix associated with the prediction on the test set;

- $\mathcal{E}$: pavement error associated with the prediction on the test set.

3 | κ-Nearest Neighbors

3.1. Fundamentals of κ-NN

κ-Nearest Neighbors (κ-NN, [17]) is a straightforward algorithm applicable to both regression and classification problems. It belongs to the category of instance-based learning or *lazy learning*, where the model doesn't explicitly learn during training. Instead, it memorizes the training instances and makes predictions based on the proximity of new data points to the existing ones. The advantage of a lazy approach lies in the flexibility to add data to the training set without re-training the model. However, this may result in slower computational times during the predictive phase. Moreover, κ-NN is a *non-parametric* method, implying that the algorithm doesn't make assumptions about the functional form of the underlying data distribution. In other words, it doesn't assume a specific mathematical model with a fixed number of parameters.

Here's a more schematic presentation of the algorithm in the context of a classification problem:

- **Training**:
  - Store all the training samples $\mathcal{D} = \{(X_i, y_i)\}_{i=1}^{N_{\text{train}}}$.
- **Prediction**:
  - Given a new data sample X_i, compute its distance from all the training points.

    Technically speaking, the codomain of a distance function should be identified by the entire $\mathbb{R}_+$ space. However, for the case at hand, we will see that the use of a function with values in $[0, 1]$ will be beneficial for the analysis. We will refer to this type of function as *dissimilarity*. This does not hinder the effectiveness of a κ-NN algorithm, and we will continue to refer to "nearest" neighbors, even if a more accurate term would be "most similar".
  - Find the κ nearest neighbors of X_i, i.e., the data points in the training set with the minimum distance from X_i.

  - Predict $\hat{y}_i$ according to some voting rule. One common rule is majority voting: the class label that is most common among the neighbors is assigned to X_i.

As evident from the above outline, when defining a κ-NN model, careful consideration must be given to the choice of the number of neighbors to take into account, the voting rule to use and the distance. In the subsequent sections, we will elucidate the reasoning behind the creation of our κ-NN model for GoOs, with a focus on the selection of these three elements.

3.2. κ-NN for GoOs

3.2.1. Number of Neighbors

To emphasize the significance of an effective choice of the parameter κ, indicating the number of nearest neighbors to take into account, we refer to Figure 3.1. This simple example illustrates that when majority voting is employed as the voting rule, different selections of κ lead to varying predictions for the outcome $\hat{y}_i$. To address this challenge, we carefully considered the selection of the number of neighbors for our specific problem. κ was included as one of the hyperparameters to be fine-tuned through 5-fold cross-validation during the training phase of our model. The chosen values for κ were 3, 5 and 7.

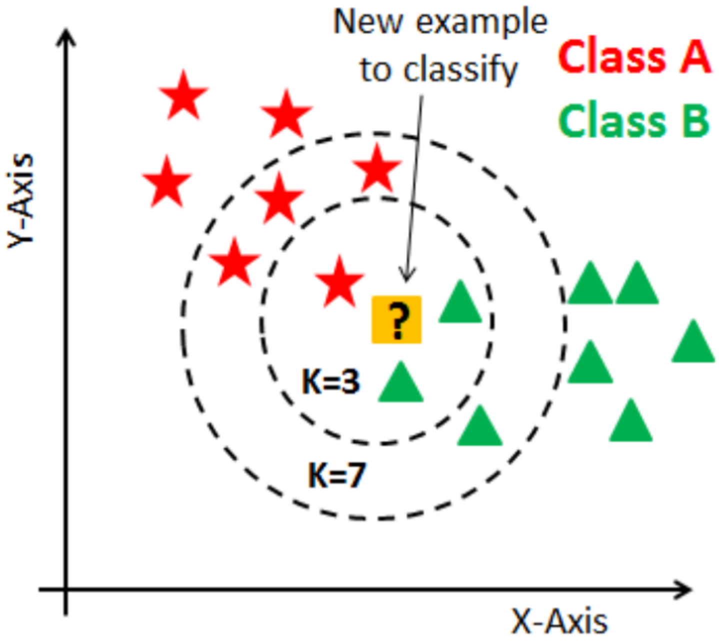

Figure 3.1: Different values of κ resulting in different predictions on a toy problem (*Source*: KDnuggets)

We specifically opted for odd values, as the majority voting rule requires an odd number of neighbors to avoid situations where the two labels have an equal number of samples among the κ nearest neighbors. It's noteworthy that, even while exploring different voting rules, we decided to retain only these three possible values for κ to streamline computational time.

3.2.2. Voting Rule

After collecting the κ nearest labeled neighbors of a data point, it was necessary to find a way to assign that data point to one of the two classes. We once again followed the work in [10] to perform this task efficiently. Firstly, we defined two vectors called $\underline{f_{\mathcal{P}}}$ and $\underline{f_{\mathcal{U}}}$, respectively indicating, for each object ω_i, the frequency of paved and unpaved neighbors out of the κ selected. Rigorously,

$$f_{\mathcal{U},i} = \frac{\sum_{j \in \mathcal{N}_i} y_j}{\kappa} \tag{3.1}$$

$$f_{\mathcal{P},i} = 1 - f_{\mathcal{U},i} \tag{3.2}$$

where $\mathcal{N}_i$ is the set of indices of the neighbors of ω_i, and $\underline{y}$ is defined according to (2.12). At this point, we introduced two hyperparameters θ_1 and θ_2 called *thresholds*, such that

$$\theta_1, \theta_2 \in [0, 1] \tag{3.3}$$

$$\theta_1 \leq \theta_2 \tag{3.4}$$

We then defined the following classification rule:

$$\hat{y}_i = \begin{cases} 0 & \text{if } f_{\mathcal{P},i} > \theta_2 \\ 1 & \text{if } f_{\mathcal{P},i} \leq \theta_1 \\ 2 & \text{otherwise} \end{cases} \tag{3.5}$$

representing the rule already anticipated in (2.7), with $\underline{f_{\mathcal{P}}}$ serving as the first column of Z.

The reader will notice that setting $\theta_1 = \theta_2 = 0.5$ achieves the classical majority voting rule. For the first part of our analysis, the set of possible values we chose to test for these hyperparameters was the one proposed in [10]: 0, 0.2, 0.4, 0.6, 0.8, 1. This choice was made for two reasons: to maintain consistency with the original work, thus allowing for an easier comparison, and to include a wide variety of possible combinations within the entire range $[0, 1]$. Initially, we considered including all values between 0 and 1 with a step of 0.1, but that led to significantly longer computations. We opted for this version with a step of 0.2, with the idea that, in case of doubt between two successive values, for example 0.4 and 0.6, the value in between, in this case 0.5, would also be tested.

3.2.3. Dissimilarity

The most intricate aspect of designing this algorithm was selecting an appropriate distance metric. In this paragraph we will introduce multiple matrices used to store information regarding distances and dissimilarities between data points. To simplify notation, we will only present the full-size versions of these matrices, which generally have N rows and $N_{\mathcal{L}}$ columns. It is important to note that, in reality, the same computations were carried out for Γ' as well, resulting in $N' \times N'_{\mathcal{L}}$ matrices. When presenting results over the limited dataset, we will not include all the apostrophes, but it is important for the reader to keep in mind the actual dimensions of the dataset used.

Energy

In [10], various established types of distances between objects were explored. The final choice was the energy distance (1.3), leading to the computation of the $N \times N_{\mathcal{L}}$ energy matrix E (1.5). This thesis aimed to generate an $N \times N_{\mathcal{L}}$ distance matrix, capturing information from both the shape of the objects Ω and the structure of the edge net H, to fully leverage the GoO structure of the dataset. With this goal in mind, we opted to retain E as a source of information on the objects and shifted our focus to the edges.

Given that many matrices introduced in the subsequent subsections consist of values in the range $[0, 1]$, we opted to scale the energy matrix as well, resulting in an energy dissimilarity measure Δ_E with values defined as follows:

$$\Delta_{E,ij} = \frac{E_{ij} - \min E}{\max E - \min E} \tag{3.6}$$

where $\min E$ and $\max E$ respectively represent the minimum and maximum values in E. The use of max and min was not the only one considered. Similar computations to the ones we will present in the next sections were also performed with a different version of Δ_E where the 0.05% and 0.95% percentiles were used for normalization. The results of this alternative computation will not be reported as they were very similar and, in general, slightly worse than the ones achieved with the normalization in (3.6).

Adjacency

Recalling the construction of the edge set, we generated the adjacency matrix A (1.8), a sparse structure containing the exact same information as H but in the form of an $N \times N$ matrix of zeros and ones. A played a dual role in the development of this κ-NN algorithm. Firstly, it was utilized to compute an $N \times N_{\mathcal{L}}$ adjacency dissimilarity matrix, assigning a dissimilarity of 1 to pairs of roads not connected to each other and 0 otherwise:

$$\Delta_{A,ij} = 1 - A_{ij} \tag{3.7}$$

Clearly, this matrix was no longer sparse due to the majority of values being equal to 1. However, only $N_{\mathcal{L}}$ columns were now necessary, as the dissimilarity between two unlabeled data points is never useful in a κ-NN. This resulted in a matrix that was not computationally prohibitive to store.

Secondly, the full sparse matrix A was also employed to execute various Graph Theory algorithms, capable of extracting information about a network starting only from its structure. The following two sections will present these algorithms and underscore their use in the computation of other dissimilarity matrices.

Shortest Paths

One intriguing distance measure between two nodes in a graph is given by the so-called *shortest path*. As the name suggests, this refers to the shortest route between two nodes in a weighted graph. In our case, with unweighted edges, this path simply consisted of the one selecting the minimum number of edges.

One of the most commonly used algorithms to compute the shortest paths from a subset of nodes to another is Dijkstra's algorithm ([13]). The simplest version of this algorithm is presented in Algorithm 3.1, with pseudocode reported from [23]. In this version of the algorithm, a single shortest path between a node *source* and a node *target* is computed. More advanced versions of this algorithm could calculate all shortest paths connecting the two nodes, but for the scope of this thesis, only one is necessary.

Dijkstra's algorithm is implemented in the `igraph` ([12]) R library. We generated an $N \times N_{\mathcal{L}}$ shortest paths matrix P, containing the outputs of the `distances` command when applied to each possible pair of objects $\omega_i \in \Omega$ and $\omega_j \in \Omega_{\mathcal{L}}$. After this, we performed the same type of normalization applied to the energy matrix and computed

Algorithm 3.1 Dijkstra's Algorithm

```
$S \leftarrow$ empty sequence
$u \leftarrow$ *target*
**if** prev[$u$] is defined **or** $u = $ *source* **then**
    **while** $u$ is defined **do**
        **insert** $u$ at the beginning of $S$
        $u \leftarrow$ prev[$u$]
    **end while**
**end if**
```

the shortest paths dissimilarity matrix:

$$\Delta_{P,ij} = \frac{P_{ij} - \min P}{\max P - \min P} \tag{3.8}$$

Community Detection

As anticipated in Chapter 2, an exceptionally valuable analysis that can be applied to graphs is the so-called community detection. This area of Graph Theory is remarkably vast, and its applications have proven to be beneficial in various fields such as epidemiology, metabolism and informatics. This paragraph provides a brief introduction to this topic, primarily based on the works in [28] and [18], and then focuses on the utility of this practice in the present case.

Intuitively, the goal of community detection is to identify small portions of a graph wherein there is a dense network of connections among elements belonging to the same community, and much sparser connections with the outside. These smaller portions are referred to as communities, hence the name of the analysis. In Figure 3.2, an example of community detection on a toy problem is presented. It is important to notice that the graph is sparse, but all communities are connected with each other as well.

The rationale behind using community detection for our GoO lies in the likelihood of a correlation between communities and neighborhoods of the city. Consequently, two objects ω_i and ω_j might not be directly connected by an edge η_{ij} while being in the same neighborhood of the city. In this case, it is not unlikely that ω_i and ω_j would share the same class of belonging.

Many different algorithms have been proposed for detecting communities in sparse graph structures starting from an adjacency matrix A. Each one of these is based on a different strategy and aims at finding a different type of hidden structure. Entering this wide world, having to decide which of these algorithms is the most reliable is an almost impossible

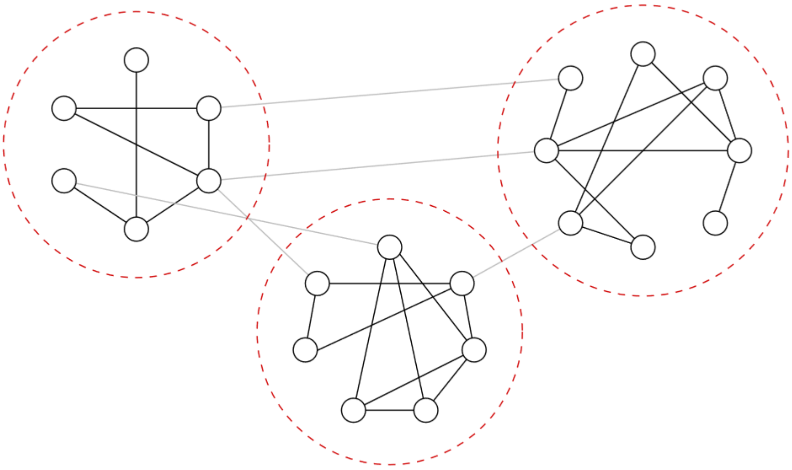

Figure 3.2: Community detection on a toy problem (*Source*: [28])

task. For this reason, we decided to select multiple algorithms already implemented in the `igraph` R library and try and use this multiple views to get a more consistent result.

We relied heavily on the work in [6] for the selection of the algorithms to use. This work defines an assessment criterion for community detection algorithms based on quality of the output and computational time. Then, it compares and contrasts all the available algorithms on different problems, both real and artificially created, providing a final ranking. While it is quite intuitive what to consider for the computational time, finding a measure for the quality of a partition is less straightforward. A commonly used measure was introduced in [11] and is called *modularity*:

$$Q = \frac{1}{2M} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2M} \right] \delta(c_i, c_j) \tag{3.9}$$

where M is the number of edges, k_i is the degree of node i (i.e. the number of other nodes to which it is connected), c_i is the assigned community and δ is a Dirac delta function returning 1 if $c_i = c_j$. A network is considered to have a valid community structure if $Q \geq 0.3$.

[6] used this definition of modularity to compute two different performance measures, one for the artificial networks ($P^{(AN)}$) and one for the real ones ($P^{(RN)}$):

$$P^{(AN)}(Q, t) = \frac{1 - (Q_d - Q)}{\text{sig}(t)} \tag{3.10}$$

$$P^{(RN)}(Q,t) = \frac{Q}{\text{sig}(t)} \tag{3.11}$$

where Q_d is the desired modularity, t is the computational time and

$$\text{sig}(t) = \frac{1}{1+e^{-\alpha t}} \tag{3.12}$$

α was empirically set to 0.1. The difference between the two performance measures lies on the fact that it is not possible to compute a desired modularity Q_d in a real world scenario, but the intuition behind is the same.

We selected the four best performing algorithms according to the ranking in [6] and applied them to our dataset. We decided to rely on the already established ranking, rather than to apply (3.11) to Γ', in order to avoid overfitting. Here we present a brief recap of the chosen algorithms, their strategy and the output on Γ'.

Fast-Greedy Modularity Optimization ([11]): This is one of the earliest proposed algorithms for community detection. It is hierarchical and aims at iteratively maximizing the modularity measure Q. Moreover, it is specifically designed for achieving high performance on sparse graphs, in contrast to [27], the other cornerstone of community detection. This makes it a suitable candidate for the case at hand. In essence, the algorithm involves identifying the changes in Q resulting from merging each pair of communities, selecting the most significant change, and executing the corresponding amalgamation.

Results on Γ':

- Number of groups: 55
- Modularity: 0.93

Multi-Level Modularity Optimization ([9]): This is another hierarchical approach iteratively maximizing Q. The main difference with the one proposed in [11] is that optimization is performed locally. Initially, individual nodes are designated as separate communities. In each step, a local, greedy reassignment occurs: nodes migrate to the community where they yield the highest contribution to modularity. Once no further reassignments are feasible, communities are consolidated, treating each as a single node. The process iterates until only one node remains or modularity can no longer be improved in a step.

Results on Γ':

- Number of groups: 59

- Modularity: 0.94

Optimization based on Random Walks ([31]): A *random walk process* (or diffusion process) on a graph is defined as follows: at each step, a walker on a node moves to a connected node by choosing randomly and uniformly among the possible neighbors.

This algorithm, generally called Walktrap, is grounded in a simple intuition: short-length random walks on a graph tend to become "trapped" within densely connected regions corresponding to communities. The authors established a measure of similarity between nodes and communities in terms of random walks of length τ, leading to the definition of a distance. This distance served as the foundation for a hierarchical clustering algorithm, where nodes are iteratively merged into progressively larger communities.

After some trials, we found that a suitable value for τ in our context was 10, resulting in low computational times and a number of subgraphs useful for our analysis.

Results on Γ':

- Number of groups: 106
- Modularity: 0.63

Optimization based on Maps of Information ([34]): This algorithm, commonly referred to as Infomap, approaches the community detection problem from a different perspective than the previous ones. Community detection is viewed as a form of data compression. The idea is to find an efficient way to code information gathered while randomly walking on a (possibly directed and weighted) graph to minimize an objective function known as *description length.* Given a partition $\mathbf{M}$ of n nodes into m communities, the description length is defined as

$$L(\mathbf{M}) = q_{\curvearrowright} H(\mathcal{Q}) + \sum_{i=1}^{m} p^i_{\circlearrowright} H(\mathcal{P}^i) \tag{3.13}$$

where $q_{\curvearrowright}$ is the probability of changing communities during a walk, $p^i_{\circlearrowright}$ is the fraction of within-community movements, and $H(\mathcal{Q})$ and $H(\mathcal{P}^i)$ represent the entropy associated with such movements.

Delving deeper into the meaning of this formulation goes beyond the scope of this thesis and would require the reader to have a deep knowledge of Coding Theory. What is important to underline is that this approach, unlike the previous ones, never focuses on modularity, and there might be cases in which an algorithm minimizing L and one maximizing Q lead to different optima. Two examples of this and reported in Figure 3.3.

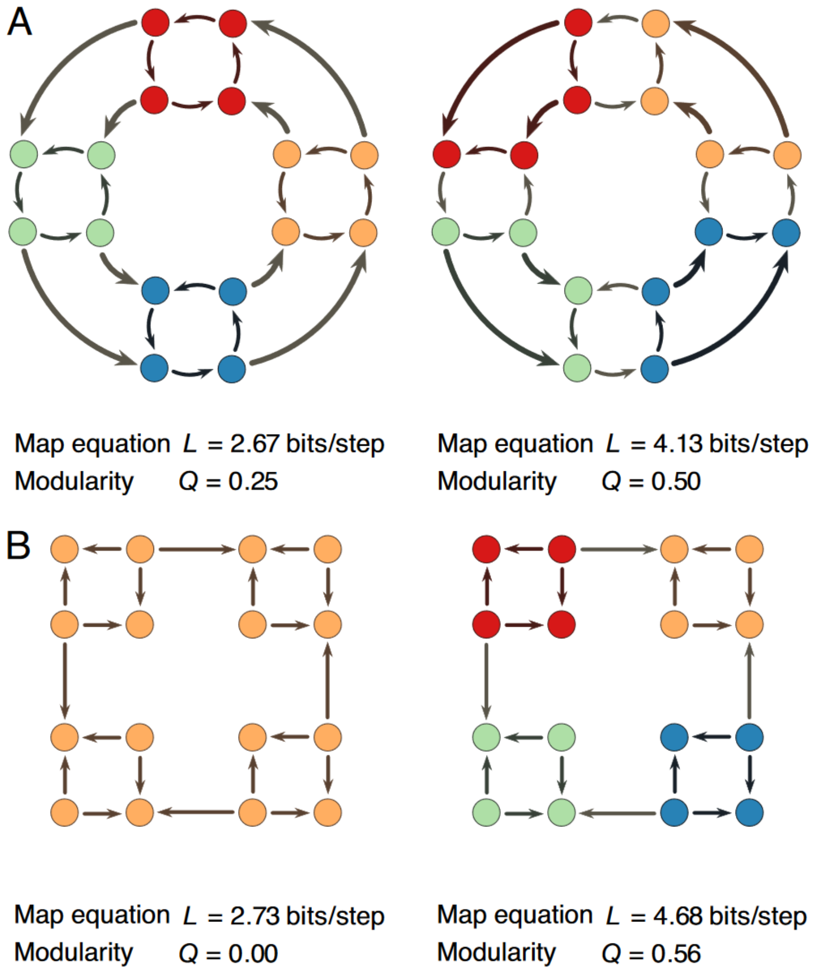
A
Map equation L = 2.67 bits/step
Modularity Q = 0.25
Map equation L = 4.13 bits/step
Modularity Q = 0.50
B
Map equation L = 2.73 bits/step
Modularity Q = 0.00
Map equation L = 4.68 bits/step
Modularity Q = 0.56

Figure 3.3: Examples of different solutions when minimizing L or maximizing Q (*Source*: [34])

We chose to incorporate this algorithm into our analysis to offer a more comprehensive view of the possible approaches to community detection, and also because the results on Γ' demonstrated optimal outcomes in terms of Q too. The primary distinction that the reader may observe is that the Infomap algorithm produces a considerably larger number of communities than Q-based algorithms. Nevertheless, the number remains sufficiently small, allowing the communities to be substantial enough to represent neighborhoods of the city.

Results on Γ':

- Number of groups: 399

- Modularity: 0.88

Now that we have presented the four community detection algorithms applied to Γ', let's explain how we incorporated this information into a dissimilarity matrix for the κ-NN. Since the goal was to assign a value to each pair of objects ω_i and ω_j, we defined the following value:

$$C_{ij} = \sum_{a \in \mathcal{A}} \delta_a(\omega_i, \omega_j) \tag{3.14}$$

where $\mathcal{A}$ is a set consisting of the four algorithms, and δ_a is a Dirac function returning 1 if $a \in \mathcal{A}$ allocates the two objects ω_i and ω_j in the same community and 0 otherwise. Intuitively, C_{ij} is a simple indicator that expresses how many times two nodes have been assigned to the same community. To ensure that it was a dissimilarity measure between 0 and 1, with low values when the two objects are part of the same community most of the time, and high values otherwise, we performed the following normalization:

$$\Delta_{C,ij} = \frac{\max C - C_{ij}}{\max C - \min C} \tag{3.15}$$

Combined Dissimilarity

In the previous paragraphs, we designed four different dissimilarity matrices, $\Delta_E, \Delta_A, \Delta_P$, and Δ_C, each encompassing values between 0 and 1 for each pair of objects ($\omega_i \in \Omega$, $\omega_j \in \Omega_{\mathcal{L}}$). At this point, we needed to find a way to construct one final dissimilarity matrix Δ, incorporating all or some of the information we just gathered, and capable of achieving the best possible classification. Numerous potential combinations of products and sums between the four original dissimilarities were possible, and we did not find anything similar in literature. We opted to explore all the combinations that made the most sense in our context, and in the next section, we will present the results for each one of those.

3.3. Implementation and Results

3.3.1. Hyperparameter Tuning

At this point, we had identified reasonable values for the number of neighbors κ, the dissimilarity Δ and the thresholds θ_1 and θ_2. The following step was to finally test our model to assess its performance. This was not a straightforward task due to the extreme complexity of the model and the large number of possible hyperparameters and resulted in two main issues. Firstly, even though we performed cross-validation, overfitting was always a risk when dealing with such a large number of possible values for the hyperparameters. Secondly, varying κ and Δ in a κ-NN model meant that the entire prediction part needed to be run for each possible combination of those parameters, leading to much longer computational times.

For these reasons, the work towards a final model was performed in multiple steps, each one refining the possible hyperparameters according to the previous results. The reader will see that, at each step, the number of values will decrease towards an optimum, while the number of possible train-test splits will be increased to ensure consistent results. The objective was to create a final model with the best possible combination of κ, θ_1, θ_2, and Δ and run that on the full dataset Γ. All the previous steps were only performed on the limited GoO Γ'.

3.3.2. Summary of the Tested Models

We wanted to find the best possible model based only on the objects and the best one based on the entire graph structure and compare the results. This would have demonstrated how our GoO structure could effectively improve the results consistently compared to a more classical Object-Oriented approach. In the following subsections, the reader will be led through all the reasoning and trials made during this phase of our work. We will also provide some insightful comments and try to explain all our choices from the coarsest versions of our models to the final ones.

Before starting, we summarise the key points distinguishing the two models for an easier understanding:

- **Object-Oriented Model ($\boldsymbol{\Omega\kappa}$-NN)**: a κ-NN model using Δ_E as dissimilarity matrix;
- **GoO-Oriented Model ($\boldsymbol{\Gamma\kappa}$-NN)**: a κ-NN model using the best combination of Δ_E, Δ_A, Δ_P and Δ_C as dissimilarity matrix.

3.3.3. Object-Oriented Model

We started with the full model and performed 5-fold cross-validation to tune the hyperparameters. We recall the possible values for such parameters:

- $\kappa = [3, 5, 7]$
- $\theta_1 = [0, 0.2, 0.4, 0.6, 0.8, 1]$
- $\theta_2 = [0, 0.2, 0.4, 0.6, 0.8, 1] \cap \theta_2 \geq \theta_1$

- $\Delta = \Delta_E$

In Table 3.1, the pavement error $\mathcal{E}$ on the test set is reported together with the chosen hyperparameters for 5 different train-test splits. We also summarize the results by computing the sample mean and sample variance of the obtained errors.

Seed	κ	θ_1	θ_2	$\mathcal{E}$
10	5	0.4	0.4	29.1%
20	5	0.4	0.4	39.3%
30	3	0.4	0.4	38.81%
40	3	0.4	0.4	42.54%
50	7	0.4	0.4	42.54%

Mean	38.46%
Var	30.43

Table 3.1: Results of the Object-Oriented κ-NN model on Γ', tuning (step 1)

It was clear that the best value for the threshold was $\theta_1 = \theta_2 = 0.4$, consistently with the results obtained in [10], so we fixed that. On the other hand, it was still not clear which was the optimal κ. This led to the choice of this new set of possible hyperparameters:

- $\kappa = [3, 5, 7]$
- $\theta_1 = 0.4$
- $\theta_2 = 0.4$
- $\Delta = \Delta_E$

Results are reported in Table 3.2.

Seed	κ	$\mathcal{E}$
60	3	39.55%
70	5	47.01%
80	3	31.72%
90	3	25%
100	5	44.03%

Mean	37.46%
Var	81.73

Table 3.2: Results of the Object-Oriented κ-NN model on Γ', tuning (step 2)

The first thing we noticed from the results at the previous step was that 7 was never chosen and even in Table 3.1 it was only chosen once. Moreover, even if 5 was selected three out of five times, the respective values of $\mathcal{E}$ were the worst on the test set, leading us to choose $\kappa = 3$. To sum up, these were the chosen hyperparameters for the final model:

- $\kappa = 3$

- $\theta_1 = 0.4$
- $\theta_2 = 0.4$
- $\Delta = \Delta_E$

In Table 3.3, we report the results for this final model.

Seed	$\mathcal{E}$
110	41.79%
120	36.94%
130	34.7%
140	34.33%
150	36.19%

Mean	36.79%
Var	8.95

Table 3.3: Results of the optimal Object-Oriented κ-NN model on Γ'

It is important to note that throughout each step of this process, we modified the five train-test splits. Readers are advised not to dwell on individual results but to focus on the average $\mathcal{E}$ and its variance. Since both of these metrics decreased while consistently reducing computational costs, we were satisfied and declared this our final Object-Oriented κ-NN model.

In Figure 3.4 we report the map of the coastal area of Maputo represented by Γ', together with our predicted values. The seed used for the split is 0.

3.3.4. GoO-Oriented Model

The first step of the design and refinement of the GoO-Oriented κ-NN was the most exploratory one. The aim was to find the best dissimilarity matrix Δ as a combination of $\Delta_E, \Delta_A, \Delta_P$, and Δ_C. As previously mentioned, the possible combinations of sums and element-wise products between these four matrices are an extremely large amount. We decided to test 20 possible dissimilarities and reported their results in Table 3.4. For the sake of clarity, for every dissimilarity matrix we only report the subscript. Moreover, sums and products have to be intended as element-wise. For instance, the voice

$$E \cdot A \tag{3.16}$$

refers to a dissimilarity Δ such that

$$\Delta_{ij} = \Delta_{E,ij} \cdot \Delta_{A,ij} \tag{3.17}$$

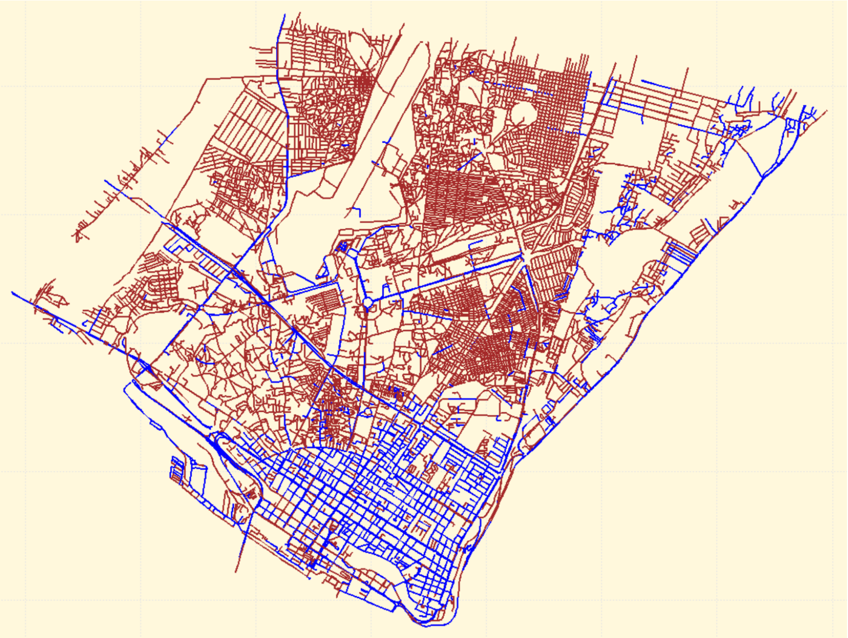

Figure 3.4: Object-Oriented κ-NN model on Γ'

🟦 Paved 🟥 Unpaved

We then decided to retain the 5 best-performing ones for further evaluation. Because of the large number of matrices, we decided to compute results on one single train-test split (seed $= 1$). In the following steps, the number of splits will increase for consistency, but for this one, we needed to limit computational time.

We recall that for all the possible dissimilarities we cross-validated and tuned the following hyperparameters:

- $\kappa = [3, 5, 7]$
- $\theta_1 = [0, 0.2, 0.4, 0.6, 0.8, 1]$
- $\theta_2 = [0, 0.2, 0.4, 0.6, 0.8, 1] \cap \theta_2 \geq \theta_1$

We noticed that one of the best results did not even include the energy dissimilarity, which was an unexpected behavior. Another good result included E as part of a product, which was uncommon and interesting at the same time. On the other hand, the two absolute bests included E in summations with one or more matrices based on edges.

At this point we wanted to find the absolute best dissimilarity. We did so by running the algorithm again with five different train-test splits. Results are reported from Table 3.5 to Table 3.9.

Δ	κ	θ_1	θ_2	$\mathcal{E}$
A	3	0.8	0.8	40.67%
P	5	0.4	0.6	**20.9**%
C	5	0.6	0.6	22.39%
$P+C$	5	0.4	0.6	24.25%
$A+P+C$	5	0.4	0.6	24.25%
$P \cdot C$	5	0.4	0.4	26.49%
$A \cdot P \cdot C$	5	0.4	0.4	26.49%
$E+A$	3	0.4	0.4	**19.4**%
$E+P$	3	0.8	0.8	**20.9**%
$E+C$	3	0.8	0.8	24.25%
$E+P+C$	3	0.8	0.8	22.39%
$E+A+P+C$	3	0.4	0.8	**19.4**%
$E+(A \cdot P \cdot C)$	3	0.8	0.8	25.75%
$(E+A) \cdot (P+C)$	5	0.4	0.6	23.88%
$E \cdot A$	5	0.4	0.4	27.61%
$E \cdot P$	3	0.4	0.4	25.75%
$E \cdot C$	5	0.4	0.6	**21.64**%
$E \cdot P \cdot C$	5	0.4	0.6	23.51%
$E \cdot A \cdot P \cdot C$	5	0.4	0.6	23.51%
$E \cdot (A+P+C)$	5	0.4	0.4	26.12%

Table 3.4: Results of the GoO-Oriented κ-NN models with different dissimilarities

The best case was the one showed in Table 3.9, which defined the dissimilarity as

$$\Delta_{ij} = \Delta_{E,ij} \cdot \Delta_{C,ij} \tag{3.18}$$

This result holds particular significance as it underscores the necessity for contributions from both the elements of Ω' and the elements from H', highlighting the strength of a GoO-based algorithm.

At this point, we knew that the most meaningful contributions were given by Δ_E and Δ_C. Moreover, multiplying the two measures gave much better results than summing them. In order to further refine our results, we added a new hyperparameter ζ that weighted the two contributions:

$$\Delta_{ij} = \Delta_{E,ij}^{\zeta} \cdot \Delta_{C,ij}^{1-\zeta} \tag{3.19}$$

Now that we had the final GoO-Oriented model, we could start with the same simulations applied for the Object-Oriented one and work towards the optimum. The starting hyperparameters were:

- $\kappa = [3, 5, 7]$

Seed	κ	θ_1	θ_2	$\mathcal{E}$
2	3	0.4	0.8	30.6%
3	5	0.6	0.6	24.63%
4	3	0.4	0.8	26.49%
5	3	0.8	0.8	24.25%
6	5	0.6	0.6	24.63%

Mean	26.12%
Var	7.04

Table 3.5: Results of the GoO-Oriented κ-NN model with $\Delta = P$

Seed	κ	θ_1	θ_2	$\mathcal{E}$
2	3	0.4	0.4	27.61%
3	3	0.4	0.4	24.63%
4	3	0.4	0.4	23.88%
5	3	0.4	0.4	25.37%
6	3	0.4	0.4	26.87%

Mean	25.67%
Var	2.39

Table 3.6: Results of the GoO-Oriented κ-NN model with $\Delta = E + A$

Seed	κ	θ_1	θ_2	$\mathcal{E}$
2	3	0.8	0.8	30.6%
3	3	0.8	0.8	27.24%
4	3	0.4	0.8	24.25%
5	5	0.6	0.6	31.72%
6	3	0.4	0.8	25.75%

Mean	27.91%
Var	10.06

Table 3.7: Results of the GoO-Oriented κ-NN model with $\Delta = E + P$

Seed	κ	θ_1	θ_2	$\mathcal{E}$
2	3	0.4	0.8	32.09%
3	3	0.8	0.8	23.88%
4	3	0.8	0.8	27.24%
5	3	0.4	0.4	17.91%
6	3	0.8	0.8	27.24%

Mean	25.67%
Var	27.39

Table 3.8: Results of the GoO-Oriented κ-NN model with $\Delta = E + A + P + C$

Seed	κ	θ_1	θ_2	$\mathcal{E}$
2	5	0.4	0.4	25.75%
3	7	0.4	0.4	23.88%
4	5	0.4	0.4	29.1%
5	5	0.4	0.4	23.88%
6	5	0.4	0.4	21.64%

Mean	24.85%
Var	7.76

Table 3.9: Results of the GoO-Oriented κ-NN model with $\Delta = E \cdot C$

- $\theta_1 = [0, 0.2, 0.4, 0.6, 0.8, 1]$
- $\theta_2 = [0, 0.2, 0.4, 0.6, 0.8, 1] \cap \theta_2 \geq \theta_1$
- $\zeta = [0.25, 0.5, 0.75]$

In Table 3.10, the pavement error $\mathcal{E}$ on the test set is reported together with the chosen hyperparameters for the same train-test splits we used while computing results in Table 3.1. This was done to allow the comparison between the two models. In further steps, we will always employ the splits tested for the Object-Oriented model.

Seed	κ	θ_1	θ_2	ζ	$\mathcal{E}$
10	5	0.4	0.4	0.25	23.88%
20	5	0.4	0.4	0.5	27.99%
30	5	0.4	0.4	0.5	20.52%
40	5	0.4	0.4	0.75	20.15%
50	5	0.4	0.4	0.25	23.13%

Mean	23.13%
Var	9.97

Table 3.10: Results of the GoO-Oriented κ-NN model on Γ', tuning (step 1)

Similarly to what happened in the previous case, this first step of tuning highlighted how the best thresholds hyperparameters were $\theta_1 = \theta_2 = 0.4$. Furthermore, this time the number of neighbors $\kappa = 5$ was also already evidently the best. A second step of tuning was still necessary in order to find the best value of ζ. The hyperparameters used were:

- $\kappa = 5$
- $\theta_1 = 0.4$
- $\theta_2 = 0.4$
- $\zeta = [0.25, 0.5, 0.75]$

Results are reported in Table 3.11.

Seed	ζ	$\mathcal{E}$
60	0.5	20.9%
70	0.5	15.3%
80	0.5	19.4%
90	0.25	20.15%
100	0.75	16.79%

Mean	18.51%
Var	5.61

Table 3.11: Results of the GoO-Oriented κ-NN model on Γ', tuning (step 2)

Taking into account both Table 3.10 and 3.11 we decided to set the optimal ζ to 0.5. This meant that the results were optimal when objects and edges of the GoO were given the same weight.

At this point, we had the final model, which used the following hyperparameters:

- $\kappa = 5$
- $\theta_1 = 0.4$
- $\theta_2 = 0.4$
- $\zeta = 0.5$

The results on Γ' are reported in Table 3.12:

Seed	$\mathcal{E}$
110	22.01%
120	25.75%
130	19.78%
140	15.67%
150	18.28%

Mean	20.3%
Var	14.61

Table 3.12: Results of the optimal GoO-Oriented κ-NN model on Γ'

The reader may notice that the mean and variance slightly increased in this final step, but this was likely caused by the use of different seeds for the splits. Nevertheless, the results remain incredibly positive, especially considering the decrease in computational time achieved by not cross-validating any hyperparameter.

Finally, in Figure 3.5 we present the same coastal area of the city, as classified by our final algorithm. The seed used is once again 0. The most evident difference with Figure 3.4 lies on the main paved road starting from south-west and going upwards for the entire map. This road is much better identified by this new approach, and some of its neighboring minor roads are realistically classified as paved too, showing the presence of the adjacency matrix in the input.

3.3.5. Choice of the Best Model

To highlight the improvement of our method with respect to the one presented in [10], we recap in Table 3.13 the performances of the Object-Oriented model, $\Omega\kappa$-NN, and the GoO-Oriented one, $\Gamma\kappa$-NN. The GoO-Oriented model dominates for every single train-test split and, for this reason, we officially declare it our final κ-NN and apply it to the

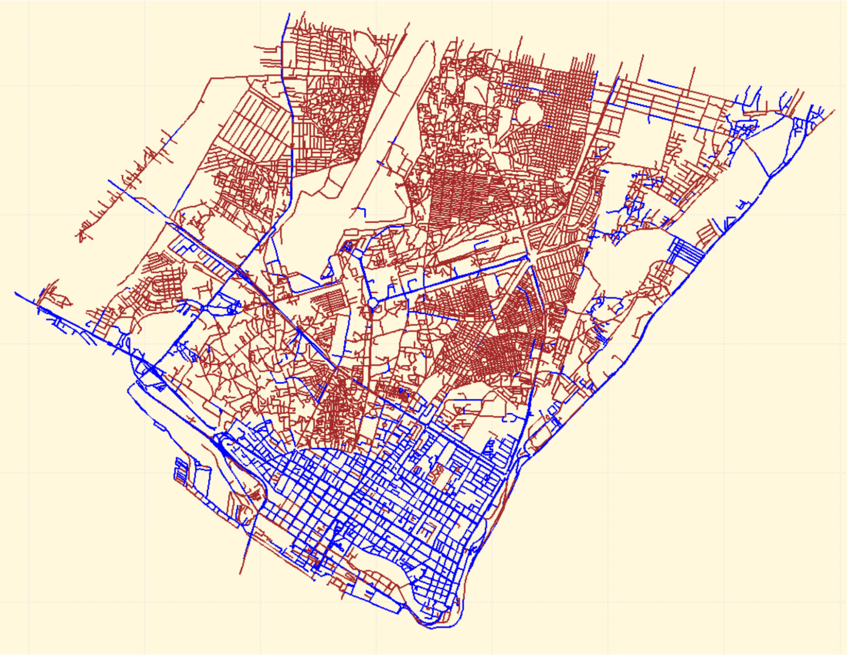

Figure 3.5: GoO-Oriented κ-NN model on Γ'

🟦 Paved 🟥 Unpaved

Seed	$\Omega\kappa$-NN	$\Gamma\kappa$-NN
110	41.79%	22.01%
120	36.94%	25.75%
130	34.7%	19.78%
140	34.33%	15.67%
150	36.19%	18.28%

Table 3.13: Comparison in terms of $\mathcal{E}$ between $\Omega\kappa$-NN and $\Gamma\kappa$-NN

entire Greater Maputo area, represented by $\Gamma(\Omega, \mathrm{H})$. In Table 3.14, we present results on various splits of the full dataset.

It is important to note that hyperparameters have been tuned on a small portion of the city, so the errors are not expected to be as optimal as in the previous case. Nevertheless, generalizing results to a larger scope allows for much shorter computational times. Further efforts in improving computational efficiency could involve parallelizing the computation of the previous sections, allowing for fine-tuning to be directly applied to the entire Greater Maputo area. This optimization would be particularly beneficial for the computation of the energy dissimilarity matrix E itself. The computation of this $N \times N_{\mathcal{L}}$ matrix was indeed time-consuming. Utilizing a computer equipped with a 10-core Intel Core i7-1265U processor and 16GB of RAM, we divided the computation of E into 20 chunks, with each chunk taking between 2.5 and 4 hours to complete.

Seed	$\mathcal{E}$
100	25.34%
200	29.11%
300	30.48%
400	35.96%
500	25%

Table 3.14: Results of the optimal GoO-Oriented κ-NN model on Γ

Finally, in Figure 3.6 we present the entire Greater Maputo area as classified by the final model. As usual, the seed used is 0.

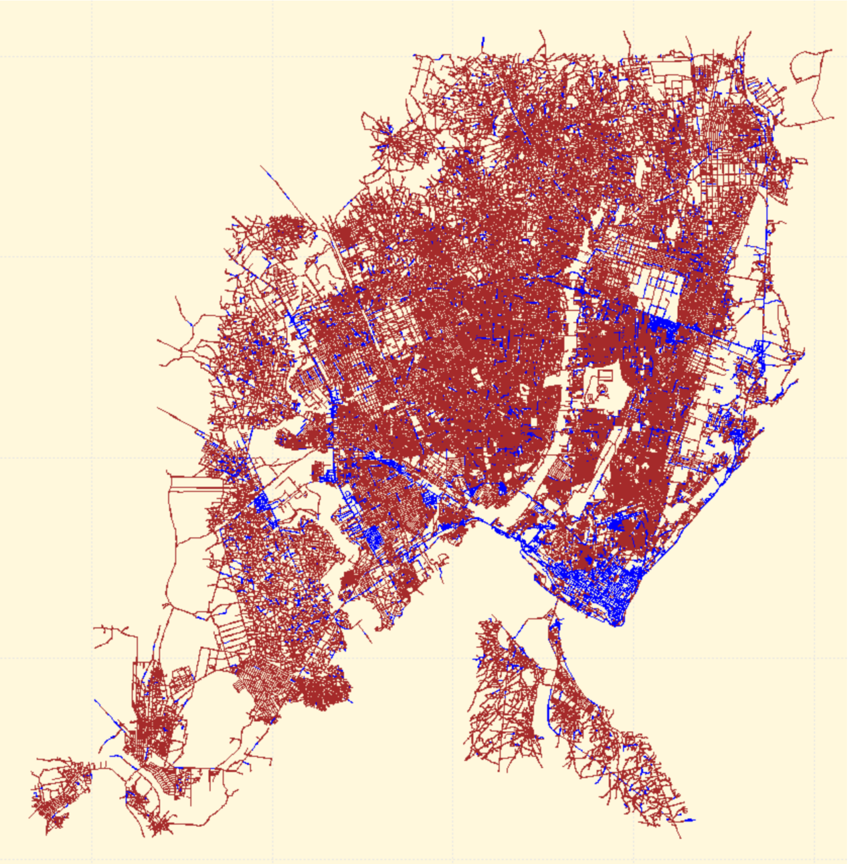

Figure 3.6: Optimal GoO-Oriented κ-NN model on Γ

🟦 Paved 🟥 Unpaved

4 | Graph Convolutional Network

4.1. Fundamentals of GCNs

4.1.1. Artifcial Neural Networks

Artificial Neural Networks (ANNs, [19], [29]) are computational models inspired by the structure and functioning of the human brain. They consist of interconnected nodes, or *artificial neurons*, organized into layers, including input, hidden and output layers. Neurons within these layers are connected by weights, which are adjusted during the learning process. An example of ANN is presented in Figure 4.1.

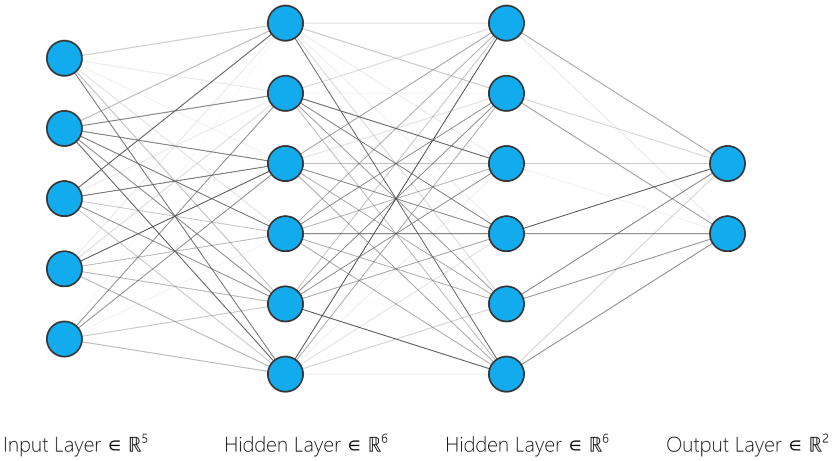
Input Layer ∈ $\mathbb{R}^5$
Hidden Layer ∈ $\mathbb{R}^6$
Hidden Layer ∈ $\mathbb{R}^6$
Output Layer ∈ $\mathbb{R}^2$

Figure 4.1: Example of ANN, where input, hidden and output layers are highlighted

The idea is the same we explained in Subsection 2.1.1. During the training of a neural network, the objective is to drive the network's output, denoted as $\hat{f}(X)$, to closely match a target function $f(X)$. The input data consists of examples of X, each accompanied by a label y_i. The output layer corresponds to the predicted output, while the intermediary layers, referred to as hidden layers, do not explicitly specify output values and their role is mainly transitory. This obscure nature of the hidden layers is the reason why neural networks are considered one of the most important examples of *black-box* models.

As previously said, every layer is made of multiple artificial neurons, each of these operating as a simplified model of its biological counterpart. The job of an artificial neuron involves processing inputs, applying weights and introducing non-linearity through an activation function. In practice, a neuron computes the dot product of the input vector X and the weight vector W, summed with a bias term b. The weights signify the strength of the connections between neurons and are crucial parameters that the network adjusts during the learning process. Following this linear combination, an activation function $a(x)$ is applied to introduce non-linearity. The Rectified Linear Unit (ReLU) is a popular activation function, defined as $f(x) = \max(0, x)$. ReLU has become widely adopted due to its simplicity and effectiveness in promoting faster convergence during training. Taking all of this into account, the output H of a neuron is computed as

$$H = a\left(\sum_{i=1}^{n} X_i \cdot W_i + b\right) \tag{4.1}$$

Training is performed by feeding the network with the same labeled data for multiple iterations, usually referred to as *epochs*. Then a process called *backpropagation* is performed. It involves the iterative adjustment of weights throughout the network to minimize the discrepancy between the predicted output $\hat{f}(X)$ and the actual target output $f(X)$. This discrepancy is typically quantified by a loss function. The most common example of loss function in classification tasks, like the one at hand, is the cross-entropy loss (2.5).

During the forward pass, the network generates predictions, and the loss is computed. Subsequently, during the backward pass, the gradient of the loss with respect to each weight is calculated. The weights are then updated in the opposite direction of the gradient through an optimization algorithm. A common example of optimization algorithm is *gradient descent*. It works by iteratively adjusting weights to minimize the loss. The gradient represents the direction of the steepest ascent in the loss landscape, and gradient descent takes steps in the opposite direction to reach a minimum. Mathematically, the weight update in gradient descent is performed as:

$$W_{\text{new}} = W_{\text{old}} - \gamma \cdot \nabla \mathcal{L}(W_{\text{old}}) \tag{4.2}$$

where γ is the learning rate, a hyperparameter that determines the size of the steps taken in the weight space.

4.1.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs, [19]) represent a specialized class of artificial neural networks designed for processing structured grid data, such as time-series (1D grids) or images (2D grids). While traditional ANNs are generally very effective, they lack the ability to capture the spatial relationships present in this type of data. CNNs address this limitation through the use of *convolutional* layers.

As the name suggests, the key innovation of CNNs lies in convolution operations. Unlike fully connected layers in traditional ANNs, convolutional layers operate on local regions of the input data using small matrices called *filters* or kernels. Each filter extracts features by sliding over the input and performing element-wise multiplication, followed by summation. This process is expressed mathematically as:

$$(X * K)_{ij} = \sum_{m} \sum_{n} X_{m,n} K_{i-m,i-n} \tag{4.3}$$

where X represents the input and K the filter. Being the filter much smaller than the input data, the entire network acquires a sparse representation. This means that we need to store fewer parameters with respect to the traditional ANN case, which both reduces the memory requirements of the model and improves its efficiency.

Pooling layers are often combined with convolutional layers to further reduce spatial dimensions and computational complexity. Max pooling is a common technique where the maximum value within a local region is retained, effectively downsampling the feature maps.

Similar to traditional ANNs, activation functions such as ReLU are applied to introduce non-linearity within CNNs. Additionally, fully connected layers may follow convolutional and pooling layers to capture global dependencies and enable high-level reasoning.

Training CNNs involves a process similar to traditional ANNs, including backpropagation and gradient descent. However, CNNs often leverage specific optimization techniques, such as stochastic gradient descent with momentum, to enhance convergence speed.

As previously stated, CNNs have found widespread application in computer vision tasks, including image classification, object detection and segmentation. Nevertheless, in the following we will mainly focus on CNNs applied to graphs for Semi-Supervised Learning. This is a more uncommon use of CNNs and it is proof of the incredibly wide possibilities of using convolutional operations within a neural network.

4.1.3. Graph Convolutional Networks

A Graph Convolutional Network (GCN, [22]) is a neural network model designed for Semi-Supervised Learning on graph-structured data. The key idea, as anticipated in Subsection 2.1.3, is to leverage the graph structure to propagate information between nodes and learn meaningful representations.

Graph-Convolutional Layers

The GCN model consists of multiple layers, each performing a graph convolution operation to update the node representations. We are not going to delve deep into the steps leading to the final convolution operation. Such steps rely on the normalized graph Laplacian, Chebyshev polynomials, and multiple approximations to make the computation faster. For an in-depth analysis of such steps, we advise reading the full work in [22]. What is fundamental to report here is the final operation performed by neurons in a graph-convolutional layer:

$$D^{-\frac{1}{2}}AD^{-\frac{1}{2}}XK = \hat{A}XK \tag{4.4}$$

where A is the adjacency matrix related to the graph, D is a diagonal matrix such that $D_{ii} = \sum_j A_{ij}$, X is the input and K the filter.

The reader may notice that notation slightly deviates from that used in [22] because, in our case, A is such that $A_{ii} = 1 \ \forall i$, eliminating the need to add an identity matrix. This decision was made to maintain coherence with the adjacency matrix A introduced for the κ-NN algorithm.

The input data is represented as an $N \times F$ matrix of features, where F denotes the number of features. Typically, this matrix comprises a set of one-hot encoded attributes. However, achieving a featureless approach is straightforward by setting $X = I$, where I is the $N \times N$ identity matrix.

Two-Layer GCN Model

The model proposed is a two-layer GCN with only one graph-convolutional hidden layer and one graph-convolutional output layer. Deeper models have been tried but performed worse because of overfitting. The activation function used for the hidden layer is the ReLU function. Since this model was invented for classifying nodes, a softmax transformation is performed in order for the output values to be probabilities that sum to 1. Suppose $\underline{v}$ is a vector in $\mathbb{R}^C$ where C is the number of possible classes, the softmax transformation is the following:

$$\text{softmax}(\underline{v})_i = \frac{e^{v_i}}{\sum_{j=1}^{C} e^{v_j}} \tag{4.5}$$

Combining all elements, the ultimate Graph Convolutional Network (GCN) proposed by [22] can be expressed mathematically as follows:

$$Z = \text{softmax}\left(\hat{A}\ \text{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right) \tag{4.6}$$

where $W^{(0)}$ and $W^{(1)}$ represent the weights fine-tuned by the neural network.

Similarly to what happens in a standard ANN, (4.6) is executed over multiple epochs on the training data X and validated using a validation set. At each iteration, the cross-entropy loss is calculated, and the weights undergo updates through gradient descent. Upon completing the final epoch, the two columns of Z correspond to the probabilities of belonging to class 0 and class 1 for the elements of the train set. Then, (4.6) with the optimized $W^{(0)}$ and $W^{(1)}$ is applied to the test set. The maximum number of epochs is capped at 200, but if the model's performance on the validation set ceases to improve for more than 5 consecutive epochs, the training phase halts automatically to prevent overfitting.

4.2. GoO Convolutional Networks

Now that we have introduced the concept of Graph Convolutional Networks, it is time to extend this idea to handle more intricate data, specifically our GoO-structured data. The initial model exploits the adjacency matrix, establishing a direct correspondence with the set of edges H; however, it lacks a mechanism for addressing objects Ω.

We devised two distinct approaches to address this challenge. The first approach sought to include meaningful information inherent in the objects through the matrix X. In contrast, the second approach dismissed X, replacing it with an identity matrix, and employed an alternative activation function for the output layer. This new activation function aimed to replace the original softmax and generate an object-adjusted Z at each epoch.

Both of these methods, which will be elucidated more comprehensively in the subsequent subsections, will continue to build upon the findings expounded in [10] and Chapter 3 of this thesis. Details about the intricate structures of the 3D objects will once again be condensed using the energy distance matrix E. Both iterations of our GoO Convolutional Networks (ΓGCNs) will endeavor to integrate this matrix into the conventional GCN framework.

4.2.1. Objects in the Input

This approach proved to be the most straightforward to implement, leveraging the inherent structure of the GCN (4.6). The primary challenge was to devise a method to incorporate information regarding each $\omega_i \in \Omega$ within an $N \times F$ matrix without compromising the distinctive 3D structure characteristic of our dataset. Having thoroughly explored the potential of a κ-NN approach involving distances between elements of Ω at this juncture, the most logical choice was to employ it again.

Once again, we relied on E, the $N \times N_{\mathcal{L}}$ energy matrix, utilizing it to compute a new $N \times (\kappa + 1)$ matrix X. This matrix was defined as the one-hot encoded version of $\underline{f_{\mathcal{U}}}$. To clarify, if $\kappa = 5$, the possible values for an element of $\underline{f_{\mathcal{U}}}$ would be [0, 0.2, 0.4, 0.6, 0.8, 1], and an object ω_i with 3 of its nearest neighbors labeled as unpaved would result in $X_i = [0, 0, 0, 1, 0, 0]$. The question may arise as to why we chose to employ E as the distance matrix for this internal κ-NN, even though we demonstrated that a GoO-based model generally yielded superior performance. This decision was influenced by the fact that the GCN was already infused with information regarding the graph's structure, specifically in the form of $\hat{A}$ in (4.6). Our intention was to avoid redundantly providing the same information, allowing for a more effective examination of the impact of objects in an edge-based algorithm. This approach marks a reversal from the strategy adopted in Chapter 3.

4.2.2. Objects in the Output

For the second approach, we decided to keep the essence of the convolutional layer (4.4) as straightforward as possible by replacing X with an identity matrix I. This choice ensures the propagation of information solely contained in the edges of the Graph of Objects. What we modified was the softmax function, which, in the original framework, transformed the network's output into a vector of probabilities. Although we aimed to maintain the output as probabilities, our objective was to incorporate information from both H and Ω. Let's define

$$B = \left[\underline{b_0}, \underline{b_1}\right] = \hat{A}\ \mathrm{ReLU}\left(\hat{A} X W^{(0)}\right) W^{(1)} \tag{4.7}$$

The novel function, termed oo-softmax (object-oriented softmax), was defined as follows:

$$\text{oo-softmax}\,(B) = \mathrm{softmax}\left(\left[\alpha \cdot \underline{b_0} + (1-\alpha) \cdot \underline{f_{\mathcal{P}}},\ \ \beta \cdot \underline{b_1} + (1-\beta) \cdot \underline{f_{\mathcal{U}}}\right]\right) \tag{4.8}$$

where, $\underline{f_{\mathcal{P}}}$ and $\underline{f_{\mathcal{U}}}$ were vectors encompassing, for each object, the frequency of paved and unpaved neighbors in terms of energy. Additionally, α and β represented two hyperparameters with values in the set $[0, 0.25, 0.5, 0.75, 1]$. This formula combined the outcomes of the computation in (4.6) and weighed their values with the results of the object-based κ-NN introduced by [10]. The rationale for choosing the object-based model over the GoO-based one remains consistent with the explanation provided in the preceding paragraph.

An interesting observation to highlight is that there was no constraint imposing the values of $\underline{b_0}$ and $\underline{b_1}$ to fall within the interval $[0, 1]$. Empirical investigations revealed that, in most instances, they did lie within the desired interval, although not invariably. Conversely, the frequencies $\underline{f_{\mathcal{P}}}$ and $\underline{f_{\mathcal{U}}}$ were, by definition, always confined to this interval. Since the essence of the formula was to compute a weighted average of these values, this initially posed a concern in our analysis. The first draft of our oo-softmax performed a sigmoid transformation of $\underline{b_0}$ and $\underline{b_1}$ before computing the weighted mean. The sigmoid function is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.9}$$

and it is straightforward to see that the output of this function is always in the desired interval. Nevertheless, as previously noted, the output values of our model were frequently situated within the desired range. Applying a sigmoid function condensed these results around 0.5, as depicted in the illustrative plot presented in Figure 4.2. This posed a more pronounced issue than before, particularly in a classification task, where values around 0.5 are evidently the least informative.

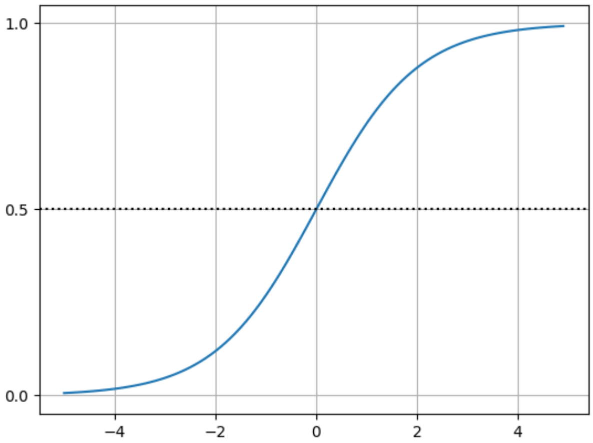

Figure 4.2: Sigmoid function $\sigma(x)$

Firstly, we tried to add a third hyperparameter $\xi > 1$, and substituted the classical sigmoid function with a steeper one: $\sigma(\xi x)$. In Figure 4.3 we present a plot of the behavior of this function when $\xi = 5$.

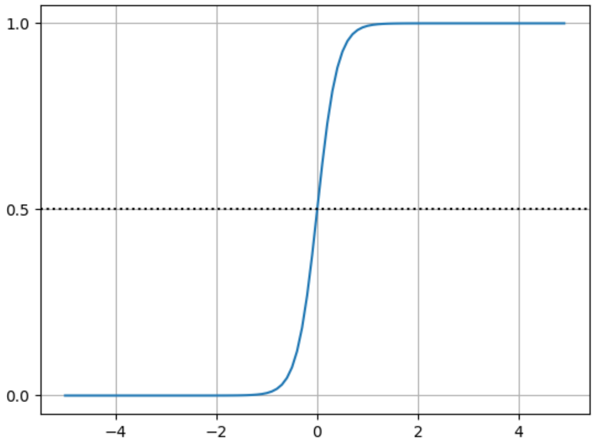

Figure 4.3: Steep sigmoid function $\sigma(5x)$

Although this second function yielded results consistent with our objectives, ultimately, we observed that retaining the original values of $\underline{b_0}$ and $\underline{b_1}$ produced outcomes that were not significantly inferior. Moreover, it allowed us to train a simpler model with one fewer hyperparameter. Consequently, we chose the simplified version in (4.8). However, if a GoO-based method were to be employed for different case studies in the future, it would be reasonable to consider incorporating this modification.

Combining everything we introduced, the final GoO-Oriented GCN model according to this second approach appears as:

$$Z = \text{oo-softmax}\left(\hat{A}\ \text{ReLU}\left(\hat{A}W^{(0)}\right)W^{(1)}\right) \tag{4.10}$$

Although the underlying intuition for this approach and the original one is the same, the two neural networks yield different results at each epoch, leading to a distinct evolution of the weights in $W^{(0)}$ and $W^{(1)}$.

4.2.3. Voting Rule

Both the original GCN and the two possible GoO-oriented modifications produce a matrix Z where each row i is a vector in $[0,1]^2$ of two probabilities that sum to 1. Similarly to what was done in Chapter 3 this lead to the possibility of trying multiple voting rules.

We introduced once again two thresholds θ_1 and θ_2 such that

$$\theta_1, \theta_2 \in \{0, 0.2, 0.4, 0.6, 0.8, 1\} \tag{4.11}$$

$$\theta_1 \leq \theta_2 \tag{4.12}$$

We then defined the following classification rule:

$$\hat{y}_i = \begin{cases} 0 & \text{if } Z_{i,1} > \theta_2 \\ 1 & \text{if } Z_{i,1} \leq \theta_1 \\ 2 & \text{otherwise} \end{cases} \tag{4.13}$$

a version of (2.7), where Z is the output of the network at the last epoch and the subscript 1 refers to the first column of Z, indicating the probability of being paved.

4.3. Implementation and Results

4.3.1. Hyperparameter Tuning

In addition to the five hyperparameters introduced in the preceding paragraphs ($\kappa, \alpha, \beta, \theta_1$, and θ_2), various other hyperparameters, characteristic of every ANN and governing its internal behavior, needed to be configured. To make the distinction between these two sets of hyperparameters more clear, we will refer to them as, respectively, GoO and ANN hyperparameters.

ANN Hyperparameters

Due to the complexity and time-consuming nature of validating all the ANN hyperparameters, we adopted the values proposed in [22] as a starting point and made empirical adjustments to determine the optimal values for improved performance. These hyperparameters remain consistent across all the algorithms presented to ensure coherent results. Several tests affirmed that the assumption of these values being optimal for all the algorithms was not far from reality.

- *Learning rate*: Set to 0.1. This is the γ in (4.2) and, as already mentioned, it can be seen as the length of the step made while updating the weights $W^{(0)}$ and $W^{(1)}$ with gradient descent.
- *Number of epochs*: Set to 200. As stressed before, this value is only the maximum number of epochs. It will be much more common for the algorithms to converge

faster.

- *Tolerance for early stopping*: Set to 5. As already mentioned, this is the number of epochs for which the performance on the validation set must remain constant for convergence to be achieved.
- *Number of neurons in hidden layer 1*: Set to 8. This is self-explanatory and is needed by the algorithm to build the first (and in our case only) hidden layer.
- *Dropout rate*: Set to 0.5. This determines the probability that each node in the GCN will be dropped out during training, which helps prevent overfitting.
- *Weight decay*: Set to $5 \cdot 10^{-4}$. When computing the error, the algorithm does not simply compute the cross-entropy loss, but also adds an L^2 regularization penalty in order to avoid overfitting. This parameter tunes the weight that is given to the regularization term.

GoO Hyperparameters

The original algorithm outlined in [22] partitioned its dataset into a training, a validation, and a test set. As previously noted, the validation set served the purpose of monitoring convergence to potentially halt training earlier, thereby preventing overfitting. This was done by checking the value of the cross-entropy loss at every epoch.

Given the introduction of additional hyperparameters to the original code, an efficient method for tuning them was required. To maintain a certain level of coherence with the κ-NN presented in the preceding chapter, we opted for 5-fold cross-validation. In each fold, 80% of the training data was utilized to fine-tune $W^{(0)}$ and $W^{(1)}$ for every conceivable combination of hyperparameters. The remaining 20% served the dual purpose of checking convergence and computing the pavement error $\mathcal{E}$ for each hyperparameter combination. We originally thought that allocating two separate validation sets, one for the convergence and the other for the assessment, was a better choice to avoid overfitting. However, that made the code much more complex and excessively decreased the dimension of the training set, without significantly improving the results. In the end, the best performing combination of hyperparameters will be selected, the model trained one last time to find the optimal $W^{(0)}$ and $W^{(1)}$ and finally used over the test set.

4.3.2. Summary of the Tested Models

We are now ready to unveil the results of the three convolutional networks. Similar to the methodology employed in Chapter 3, we will determine the optimal set of hyperparameters for each network, this time focusing on the GoO hyperparameters. Subsequently, we will conduct a comparative analysis, emphasizing the performance differences and highlighting the superiority of a GoO-Oriented model over an Edge-Oriented one.

Differently from what was done for the κ-NN, this time we will present two distinct GoO-based algorithms, as introduced in Section 4.2. In this manner, we will not only demonstrate the necessity of including objects in the analysis but also elucidate the optimal way to do so.

Before starting, we summarise the key points distinguishing the three models for an easier understanding:

- **Edge-Oriented Model (HGCN)**: a GCN performing (4.6) with $X = I$;
- **First GoO-Oriented Model ($\boldsymbol{\Gamma}$GCN$_1$)**: a GCN performing (4.6) with X equal to the one-hot encoded version of $\underline{f_{\mathcal{U}}}$;
- **Second GoO-Oriented Model ($\boldsymbol{\Gamma}$GCN$_2$)**: a GCN performing (4.10).

4.3.3. Edge-Oriented Model

The first model we present is the original GCN proposed in [22] with $X = I$, so that no information about the objects is given. The only hyperparameters to tune were the thresholds and the possible values were:

- $\theta_1 = [0, 0.2, 0.4, 0.6, 0.8, 1]$
- $\theta_2 = [0, 0.2, 0.4, 0.6, 0.8, 1] \cap \theta_2 \geq \theta_1$

In Table 4.1, the pavement error $\mathcal{E}$ on the test set is reported together with the chosen hyperparameters for 5 different train-test splits. We also summarize the results by computing the sample mean and sample variance of the obtained errors.

Seed	θ_1	θ_2	$\mathcal{E}$
10	0.4	0.4	31.34%
20	0.4	0.4	22.01%
30	0.6	0.6	38.81%
40	0.4	0.4	49.25%
50	0.4	0.6	60.45%

Mean	40.37%
Var	225.78

Table 4.1: Results of the Edge-Oriented GCN model on Γ', tuning (step 1)

It was clear that the variability of the results posed a main concern on the generalizability of this approach. However, we noticed that, differently from what happened with the κ-

NN, the best set of thresholds was not really clear. Both 0.4 and 0.6 were chosen multiple times, so better results may have lied in the middle. We updated the set of possible hyperparameters to:

- $\theta_1 = [0.4, 0.5, 0.6]$
- $\theta_2 = [0.4, 0.5, 0.6] \cap \theta_2 \geq \theta_1$

and simulated the problem again over five new splits. Results are in Table 4.2

Seed	θ_1	θ_2	$\mathcal{E}$
60	0.5	0.5	25.0%
70	0.5	0.5	23.88%
80	0.5	0.5	22.39%
90	0.5	0.6	23.88%
100	0.5	0.5	30.22%

Mean	25.07%
Var	9.13

Table 4.2: Results of the Edge-Oriented GCN model on Γ', tuning (step 2)

The improvement achieved by this simple modification was dramatic so we decided to set our optimal hyperparameters:

- $\theta_1 = 0.5$
- $\theta_2 = 0.5$

Results of some final simulations are reported in Table 4.3.

Seed	$\mathcal{E}$
110	24.63%
120	31.34%
130	28.36%
140	21.27%
150	21.64%

Mean	25.45%
Var	18.95

Table 4.3: Results of the optimal Edge-Oriented GCN model on Γ'

These results align with those obtained through cross-validation of the thresholds and are highly satisfactory. This new algorithm proves to be significantly more efficient than the Object-Oriented κ-NN, highlighting the impact of the edges of the GoO on the final classification.

To conclude our work on the Edge-Oriented GCN model, in Figure 4.4 we present a plot of our results over the coastal area of Maputo represented by Γ'. One again, the seed used for the train-test split is 0.

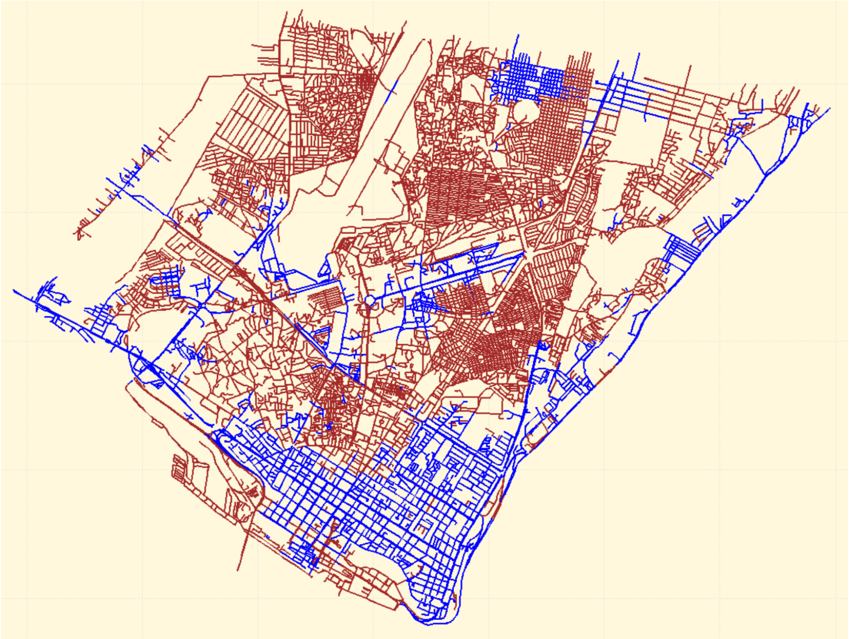

Figure 4.4: Edge-Oriented GCN model on Γ'

🟦 Paved 🟥 Unpaved

4.3.4. First GoO-Oriented Model

We proceeded with the same approach for the first GoO-based model, this time using as X the one-hot encoded version of $\underline{f_{\mathcal{U}}}$. The initial possible values for the hyperparameters were:

- $\kappa = [3, 5, 7]$
- $\theta_1 = [0, 0.2, 0.4, 0.6, 0.8, 1]$
- $\theta_2 = [0, 0.2, 0.4, 0.6, 0.8, 1] \cap \theta_2 \geq \theta_1$

Results are reported in Table 4.3.

Seed	κ	θ_1	θ_2	$\mathcal{E}$
10	5	0.6	0.6	24.25%
20	3	0.8	0.8	24.25%
30	3	0.8	0.8	29.48%
40	5	0.6	0.6	35.82%
50	7	0.6	0.6	26.12%

Mean	27.98%
Var	23.75

Table 4.4: Results of the first GoO-Oriented GCN model on Γ', tuning (step 1)

The obtained results did not directly lead to one single set of optimal hyperparameters, so further tuning was needed. Nevertheless, we managed to understand that the best values for θ_1 and θ_2 lied between 0.6 and 0.8. We took this information into account and ran the model again with the following possible parameters:

- $\kappa = [3, 5, 7]$
- $\theta_1 \in [0.6, 0.7, 0.8]$
- $\theta_2 \in [0.6, 0.7, 0.8] \cap \theta_2 \geq \theta_1$

The results are presented in Table 4.5.

Seed	κ	θ_1	θ_2	$\mathcal{E}$
60	3	0.6	0.7	33.96%
70	5	0.6	0.6	16.04%
80	5	0.7	0.7	17.91%
90	3	0.6	0.6	18.66%
100	3	0.6	0.7	23.13%

Mean	21.94%
Var	51.93

Table 4.5: Results of the first GoO-Oriented GCN model on Γ', tuning (step 2)

Once again, there was no clear optimal set of hyperparameters. However, by taking into account both results in Table 4.4 and Table 4.5, we decided to set them to:

- $\kappa = 5$
- $\theta_1 = 0.6$
- $\theta_2 = 0.6$

At this point, we proceeded with the training of the optimal model over the same five train-test splits we used for the best Edge-Oriented GCN, for later comparison. Results are reported in Table 4.6.

Seed	$\mathcal{E}$
110	17.16%
120	30.97%
130	30.22%
140	20.9%
150	17.54%

Mean	23.36%
Var	45.83

Table 4.6: Results of the first optimal GoO-Oriented GCN model on Γ'

In Figure 4.5, the plot of the results over the coastal area of Maputo is presented. Notably, some of the northern roads, which were curiously classified as paved in Figure 4.4, are now categorized as unpaved, a result we deem more realistic.

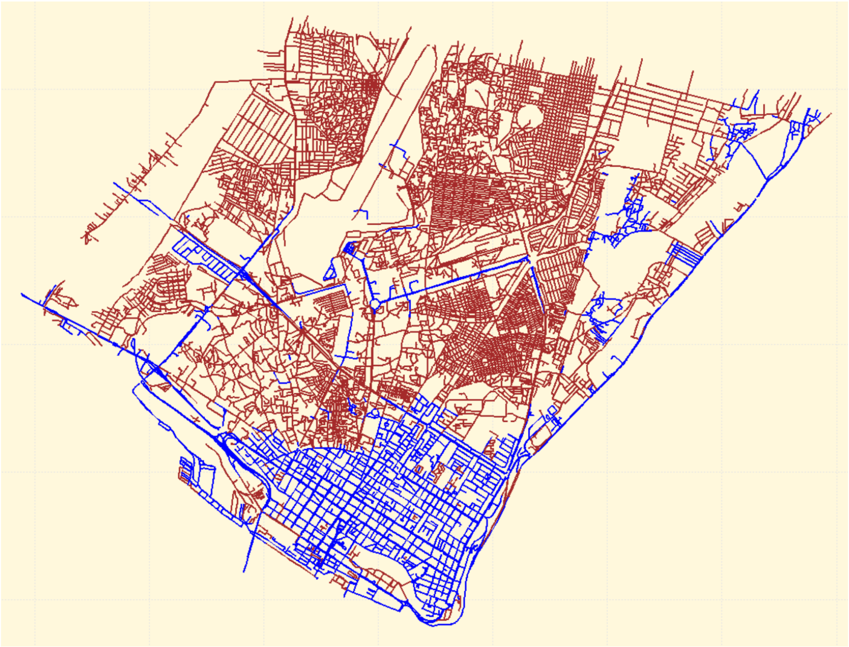

Figure 4.5: First GoO-Oriented GCN model on Γ'

🟦 Paved 🟥 Unpaved

4.3.5. Second GoO-Oriented Model

At this point, the only thing left to do was to find the optimal hyperparameters for our second proposed GoO-Oriented GCN. Schematically, the starting sets of values for these parameters were:

- $\kappa = [3, 5, 7]$
- $\alpha = [0, 0.25, 0.5, 0.75, 1]$
- $\beta = [0, 0.25, 0.5, 0.75, 1]$
- $\theta_1 = [0, 0.2, 0.4, 0.6, 0.8, 1]$
- $\theta_2 = [0, 0.2, 0.4, 0.6, 0.8, 1] \cap \theta_2 \geq \theta_1$

Computations at this stage of our work were obviously the longest of the entire work, given the large amount of possible hyperparameters combinations. This was where the use of a limited dataset $\Gamma'(\Omega', \mathrm{H}')$ really proved to be a winning choice. Results of this step are in Table 4.7.

From this we realised that once again the optimal thresholds were $\theta_1 = \theta_2 = 0.6$, while the best number of neighbors was $\kappa = 3$. Unfortunately, the choice of α and β was not as straightforward. However, we discovered that the best results were obtained when $\alpha < \beta$

Seed	κ	α	β	θ_1	θ_2	$\mathcal{E}$
10	5	0	0.5	0.6	0.6	20.52%
20	3	0	0.5	0.6	0.6	18.28%
30	3	0.25	0.5	0.6	0.6	24.25%
40	3	0.25	0.5	0.6	0.6	32.46%
50	3	0	0.75	0.6	0.6	15.3%

Mean	22.16%
Var	43.81

Table 4.7: Results of the second GoO-Oriented GCN model on Γ', tuning (step 1)

and $\beta < 1$. Keeping this in mind, we limited the range of possible values to:

- $\kappa = 3$
- $\alpha = [0, 0.25]$
- $\beta = [0.25, 0.5, 0.75]$
- $\theta_1 = 0.6$
- $\theta_2 = 0.6$

New tests were made and their results are reported in Table 4.8.

Seed	α	β	$\mathcal{E}$
60	0	0.75	26.87%
70	0.25	0.75	16.79%
80	0	0.75	18.66%
90	0	0.5	19.03%
100	0	0.5	21.27%

Mean	20.52%
Var	15.12

Table 4.8: Results of the second GoO-Oriented GCN model on Γ', tuning (step 2)

At this point, we immediately set $\alpha = 0$, while for β a little more reasoning was needed. Both 0.5 and 0.75 were chosen multiple times, but the worst result was obtained when $\beta = 0.75$. For this reason, we opted for $\beta = 0.5$ as the final value. To recap:

- $\kappa = 3$
- $\alpha = 0$
- $\beta = 0.5$
- $\theta_1 = 0.6$
- $\theta_2 = 0.6$

The results on Γ' of the second optimal GoO-Oriented GCN are reported in Table 4.9, while the usual plot with seed equal to 0 is in Figure 4.6. A visual inspection of the classified map reveals a results that is in between the two previous models. Some roads in the northern part are classified as paved, differently from Figure 4.5, but their structure appears to be less random and more realistic than the one visible in Figure 4.4.

Seed	$\mathcal{E}$
110	26.12%
120	25%
130	22.76%
140	15.67%
150	16.04%

Mean	21.64%
Var	18.66

Table 4.9: Results of the second optimal GoO-Oriented GCN model on Γ'

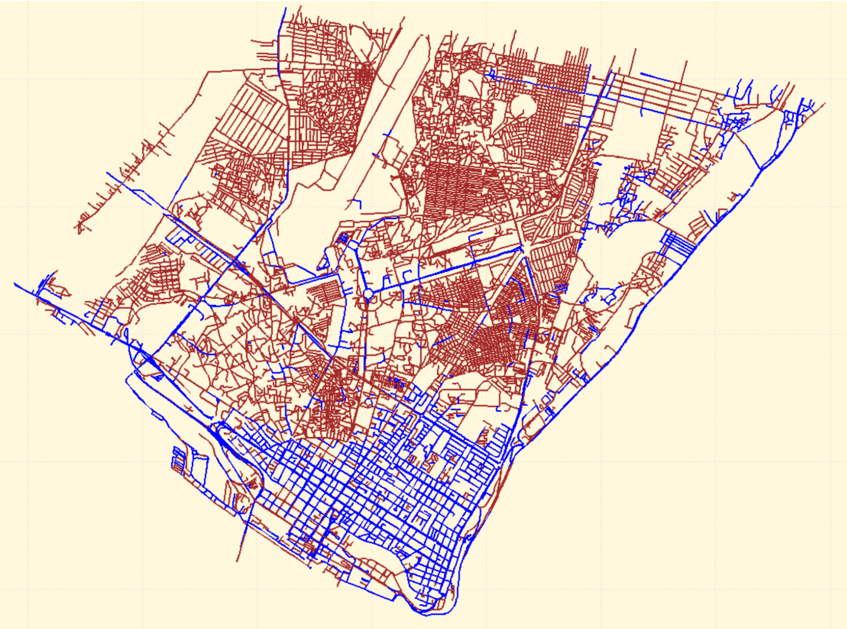

Figure 4.6: Second GoO-Oriented GCN model on Γ'

🟦 Paved 🟥 Unpaved

4.3.6. Choice of the Best Model

Similarly to what was done in Chapter 3, we recap our findings and summarise the results of our different GCNs in Table 4.10 for a final comparison.

On average, both our methods outperformed the original Edge-Oriented neural network, highlighting not only the absolute superiority of a GoO-structured dataset, but also that our models effectively exploited information coming both from Ω and H. The second method generally demonstrated superior performance compared to the first one, present-

Seed	HGCN	ΓGCN$_1$	ΓGCN$_2$
110	24.63%	17.16%	26.12%
120	31.34%	30.97%	25%
130	28.36%	30.22%	22.76%
140	21.27%	20.9%	15.67%
150	21.64%	17.54%	16.04%

Table 4.10: Comparison in terms of $\mathcal{E}$ between HGCN, ΓGCN$_1$ and ΓGCN$_2$

ing lower mean and variability. Nevertheless, it's worth noting that one of the splits produced an error even larger than that obtained by HGCN, so the choice for a final optimal model was not straightforward. However, to facilitate a comprehensive comparison against the $\Gamma\kappa$-NN presented at the end of Chapter 3, we wanted to choose one single final model, so we opted for ΓGCN$_2$ anyway.

To conclude the chapter, we present the performance of the chosen model over the entire Greater Maputo area. In Table 4.11 results for 5 different train-test splits of $\Gamma(\Omega, \mathrm{H})$ can be found.

Seed	$\mathcal{E}$
100	31.16%
200	38.7%
300	35.45%
400	34.93%
500	41.78%

Table 4.11: Results of the optimal GoO-Oriented GCN model on Γ

Similarly to what was said for the κ-NN, these results may appear disappointing with respect to the ones obtained on the reduced GoO. It must not be forgotten that this is the optimal model on Γ' and the results in Table 4.11 must be simply seen as a way to assess the level of generalizability of such specific model. In this perspective, results are indeed satisfying, taking into account the extreme shrink in terms of computational time this process allowed.

Figure 4.7 presents a plot of the area with the classification performed. As usual, the seed used in this final case is 0.

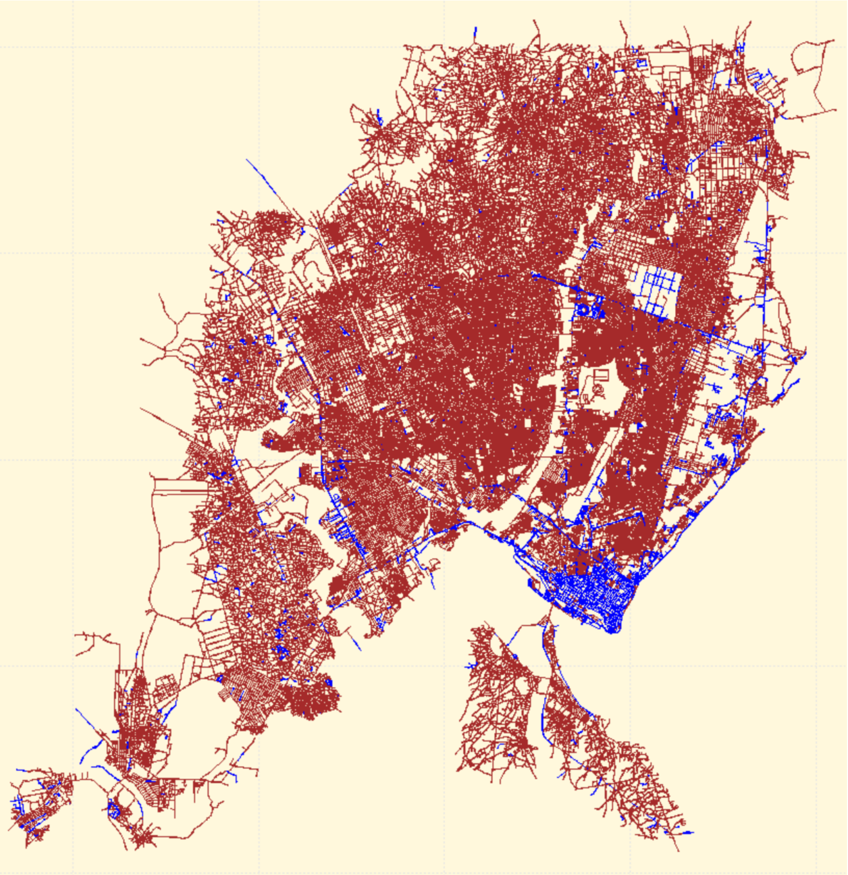

Figure 4.7: Optimal GoO-Oriented GCN model on Γ

🟦 Paved 🟥 Unpaved

5 | Compared Approaches

5.1. Comparison

The aim of this concluding chapter is to conduct a comparative analysis between the two approaches presented in Chapter 3 and Chapter 4 to discern if one consistently outperforms the other. To achieve this, our focus will be solely on the two successful models, $\Gamma\kappa$-NN and ΓGCN_2 (from now on, simply ΓGCN), executed with optimal hyperparameters. We will refrain from conducting additional cross-validation or optimization. As a reference, we will solely rely on the standard sets of five distinct train-test splits. It is essential to reiterate that these splits were meticulously crafted to ensure that identical training and test sets were employed for both the κ-NN and the neural network. This approach facilitates a more impartial comparison of performance.

In the following subsections, we will analyze four key characteristics that we deemed essential during our investigation:

- **Performance**: mean and variance of the pavement error $\mathcal{E}$ over the limited Graph of Objects Γ', utilizing the optimal set of hyperparameters;
- **Generalizability**: mean and variance of the pavement error $\mathcal{E}$ over the entire Graph of Objects Γ, employing the optimal hyperparameters found for Γ';
- **Computational Time**: time required by the algorithm to execute the classification of all the roads in Γ' and Γ once the optimal hyperparameters have been determined;
- **Visual Result**: a posteriori examination of the plot of the classified roads to identify any areas of the city exhibiting peculiar or unexpected behaviors. Additionally, the actual structure of the city is taken into account.

Considering all of these aspects, we will ultimately determine whether one of the two approaches objectively outperforms the other.

5.1.1. Performance

For this first point we report in Table 5.1 the results computed in the previous chapters for the optimal models over five different train-test splits, together with their mean and variance. For a faster analysis, the best result for each seed is written in bold.

Seed	$\Gamma\kappa$-NN	ΓGCN
110	**22.01%**	26.12%
120	25.75%	**25%**
130	**19.78%**	22.76%
140	**15.67%**	**15.67%**
150	18.28%	**16.04%**
Mean	20.3%	21.64%
Var	14.61	18.66

Table 5.1: Compared performance of the two best models

Out of the five train-test splits, both algorithms outperformed each other twice, while for seed equal to 140, they had the exact same performance. The mean and variance of the $\Gamma\kappa$-NN are slightly lower, but the difference was not significant enough to be considered statistically significant. Hence, it was not possible to definitively determine a best model between the two.

From a practical standpoint, having two competitive algorithms provides practitioners and decision-makers with valuable flexibility. Depending on the specific requirements of a task or the nature of the dataset, they can choose between $\Gamma\kappa$-NN and ΓGCN, confident in the knowledge that both options are capable and dependable.

5.1.2. Generalizability

Now we report the results obtained over Γ by $\Gamma\kappa$-NN and ΓGCN, to see if the two algorithms scale with the same efficiency as well. In Table 5.2 the pavement error over five splits can be found, along with mean and variance for the two algorithms.

It is clear that the situation is different from the one presented in the previous subsection. In this case, $\Gamma\kappa$-NN outperforms ΓGCN four out of five times, with a mean pavement error $\mathcal{E}$ that is around 7% inferior. Variance of the second method is slightly lower, but this is not enough to make it a better model. It appears that, even if the two models are basically equivalent at classifying roads in the limited dataset, the first one scales much better. One possible reason for such a discrepancy might simply be that the optimal hyperparameters found for the limited coastal area of the city are significantly different

Seed	$\Gamma\kappa$-NN	ΓGCN
100	**25.34%**	31.16%
200	**29.11%**	38.7%
300	**30.48%**	35.45%
400	35.96%	**34.93%**
500	**25%**	41.78%
Mean	29.18%	36.4%
Var	19.97	16.17

Table 5.2: Compared generalizability of the two best models

than the best ones over the entire Greater Maputo area. On the other hand, the issue might be in the algorithm itself, performing worse with an increasing size of the dataset. This unforeseen aspect enriches our exploration, opening avenues for further investigation into the underlying mechanisms that contribute to the scalability of statistical models in this context.

5.1.3. Computational Time

For this third point, we focused on the computational time needed for the two algorithms to perform the classifications in Table 5.1 and Table 5.2. The time interval taken into account for this analysis was from the computation of $\underline{f_{\mathcal{U}}}$ (from Δ for $\Gamma\kappa$-NN and from E for ΓGCN) to the final calculation of $\mathcal{E}$ over the test set. The time for the computation of A, E and C was not considered, since it was enough to do it once a priori. Still, it should be taken into account that the computation of C was not needed for the neural network, making it more efficient from that point of view. The time for the upload of such matrices (which was significant especially for the prediction over the entire Γ) was also not considered, as it was not depending on the algorithms but on the language used. Since R was used to write the κ-NN and Python was used for the GCN, taking this part of the computation into account would have biased the results. The simulations where run on a laptop equipped with a 10-core Intel Core i7-1265U processor and 16GB of RAM.

In Table 5.3 and Table 5.4 the computational times over Γ' and Γ are reported for every train-test split. It is crucial to emphasize that, in contrast to the pavement errors computed in the previous subsections, the computational times may vary slightly each time an algorithm is executed. Therefore, the best performing mean was written in bold, rather than the best value for each seed.

Undoubtedly, for a limited dataset, the κ-NN outperforms the ΓGCN, requiring, on average, less than half the time for classification. However, as the number of data points in-

Seed	$\Gamma\kappa$-NN	ΓGCN
110	0.86 s	2.07 s
120	0.86 s	2.59 s
130	0.83 s	2.15 s
140	0.86 s	2.08 s
150	0.83 s	2.62 s
Mean	**0.85 s**	2.3 s

Table 5.3: Compared computational time of the two best models over Γ'

Seed	$\Gamma\kappa$-NN	ΓGCN
100	28.72 s	20.75 s
200	37.91 s	19.35 s
300	38.53 s	19.79 s
400	41.59 s	25.81 s
500	49.23 s	19.41 s
Mean	39.2 s	**21.02 s**

Table 5.4: Compared computational time of the two best models over Γ

creases, the neural network demonstrates significantly faster performance. Consequently, the second approach exhibits less favorable scalability in terms of performance but fares considerably better in terms of computational time.

Given that, when focusing on the limited dataset, the choice is between an algorithm that takes slightly less than a second and one that takes around two seconds (both incredibly low time intervals) we deemed it more meaningful to prioritize the results over the entire GoO. Therefore, in terms of computational times, the neural network emerges as a much more efficient approach.

5.1.4. Visual Result

As the primary objective of this thesis was to address a classification task in a real-world scenario, we deemed it essential to conclude our work with an exploratory analysis of the results within this context. In practice, this entailed a comprehensive study of the plot and a visual examination of areas where the behavior of paved and unpaved roads appeared unexpected or unusual. This analysis will be specifically conducted over Γ', primarily due to the impracticality of visually scrutinizing the entire road network, which comprises over 53000 data points. In Figure 5.1, the plots of the results over Γ' with seed equal to 0 are once again presented.

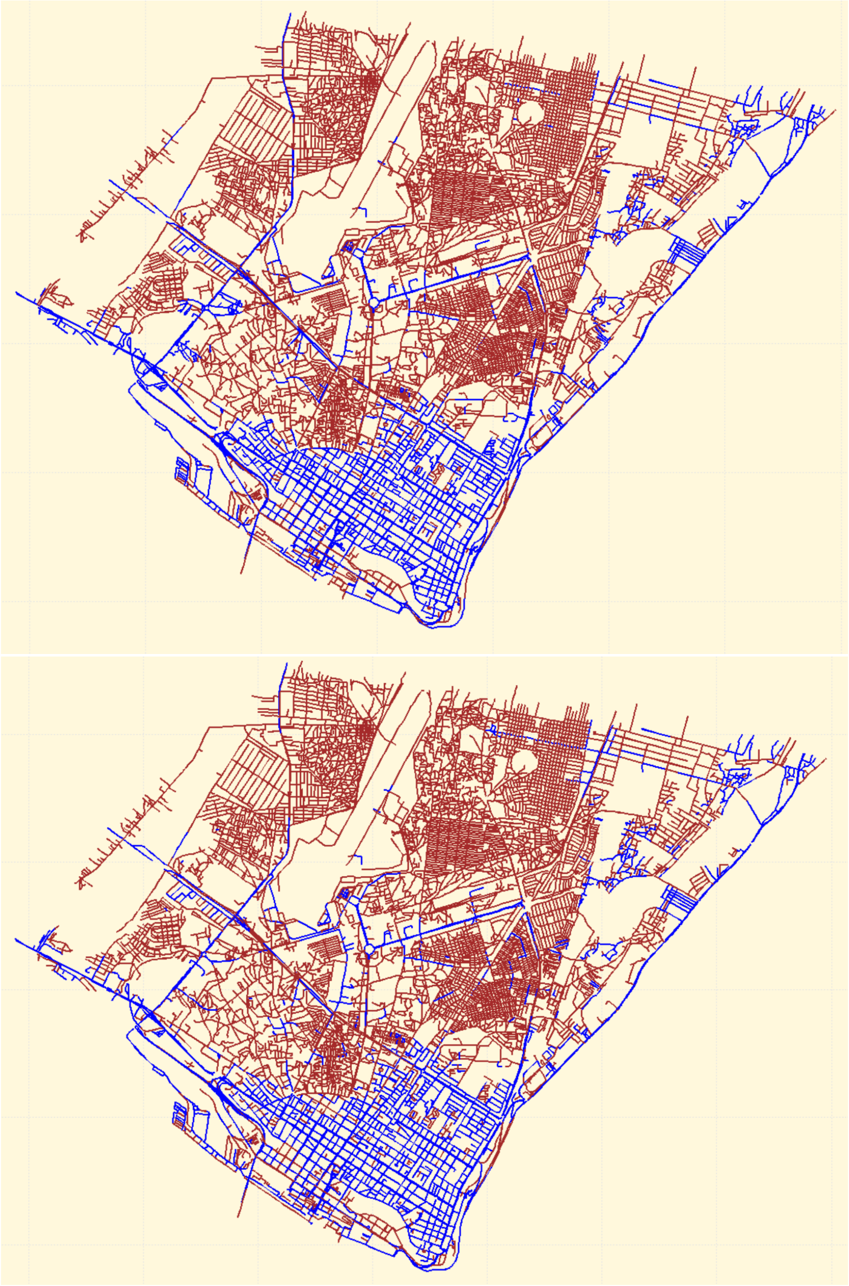

Figure 5.1: Visual results over Γ' for $\Gamma\kappa$-NN (top) and ΓGCN (bottom)

It is not surprising to observe that the overall characteristics of the two predicted road networks are quite similar. Both algorithms have demonstrated remarkable reliability and exhibited strong performances in this specific area of the city. In both cases, the region immediately adjacent to the coast (southeastern part of the map) features a substantial number of paved roads, indicative of a densely populated and vibrant neighborhood. As one moves away from this area, the prevalence of short unpaved roads increases exponentially, pointing towards the suburbs.

Some minor differences are present. For instance, the primary paved road originating from the south-west and traversing the suburbs was better identified by the κ-NN. Conversely, the region in the north-east exhibits a more coherent structure in the second picture. Nevertheless, these slight discrepancies are certainly not substantial enough to declare one of the two algorithms superior to the other.

5.2. Discussion

Taking into account all aspects analyzed in this chapter, the selection of a single best algorithm is not straightforward, and perhaps even impossible. We emphasized how the two models yield remarkably similar results when applied to the limited dataset with tuned hyperparameters. These results are highly satisfying in terms of accuracy and pavement error. Either of the two algorithms could be confidently reused if a similar analysis had to be conducted in the future with other Graphs of Objects of comparable dimensions. The quality of this performance was not only assessed mathematically but was also confirmed through visual analysis of the results. On the other hand, as the dimension of the dataset increases exponentially, the first algorithm begins to outperform the second one, albeit at a progressively slower pace. The choice between these two characteristics (performance and computational speed) will ultimately depend on the user's priorities for their analysis.

In light of the various considerations we have enumerated, it becomes evident that a singular, optimal algorithm for the comprehensive analysis of GoOs cannot be conclusively determined. This holds true not only for the broader context of GoO-based analysis but also specifically in the context of our current research. Within the scope of our investigation, we have presented two distinct models, each exhibiting commendable performance metrics, yet accompanied by its own set of strengths and limitations. Consequently, the ultimate decision on which model to employ rests upon a nuanced evaluation of the unique demands and objectives of the analytical task at hand.

Conclusion

The issues related to unreliable and unsafe road networks, especially in developing countries, are vast and still widely open. Nevertheless, the work presented in this thesis represents one step further towards the accomplishment of the Sustainable Development Goals, designed by the United Nations in 2015.

We have explained the need for an accurate classification of the pavement status of roads in big cities, with a specific focus on the Greater Maputo area in Mozambique. After having visualised this goal, we focused on a way to efficiently represent data in this context. Working with satellite images of a road network is indeed a really difficult task, that could be faced in a vast variety of ways. In our case, we decided to represent these images as 3D objects in the RGB space and to interconnect such objects in a graph, where each border between roads assumed the role of an edge. We called this new data structure Graph of Objects.

Afterwards, we needed to prove that this type of representation was actually useful for the classification. We decided to start from two extremely different already existing algorithms, a κ-NN and a Graph Convolutional Network. The former was originally designed to work with an object-oriented dataset, while the latter was designed to gain information from the edge structure of a graph. Our objective was to evolve these algorithms so that they might work on our Graph of Objects, combing the realms of Object-Oriented Data Analysis and Graph Theory. The two final algorithms we produced proved to perform much better than their original counterparts, confirming the big step forward represented by GoO-structured information.

In a final chapter, we compared our two models according to multiple performance metrics, concluding that none of them is better than the other, but the choice should depend on the dimension of the dataset at hand and the available time. This analysis also represented a contrast between the worlds of traditional Statistical Learning and Deep Learning, showcasing how both approaches may be able to utilize this new type of data structure in different, but always well-performing, ways.

Future work may focus on Graphs of Objects in completely different contexts, varying from medicine to world wide web analysis. It is quite common to find problems where data cannot be represented in the most traditional ways and a GoO structure may be useful every time data points are complex and connected with each other. New algorithms may be developed and placed side by side with our $\Gamma\kappa$-NN and ΓGCN. This is just the beginning for GoO-Oriented analysis.

A | Guide to the Code

In this appendix, we provide a guide to the code that was implemented for the computations carried out throughout the thesis. For a comprehensive understanding, it is recommended to download the code available at `https://github.com/bertrandpouget/goo`.

Data

The dataset is supposed to be added by the user in a folder called `data` and should contain the input presented in Section 2.5. More specifically, the folder should contain two subfolders, `python` and `r` composed like this:

- `python`
  - `e`: `.txt` file containing the $N \times N_{\mathcal{L}}$ energy matrix after the normalization (3.6);
  - `a`: `.mtx` file containing the full $N \times N$ adjacency matrix encoded as a sparse matrix;
  - `y`: `.txt` file containing the one-hot-encoded $N \times 2$ response variable defined according to

    $$\mathrm{y}_i = \begin{cases} [1 \quad 0] & \text{if } \omega_i \text{ is paved} \\ [0 \quad 1] & \text{if } \omega_i \text{ is unpaved} \\ [0 \quad 0] & \text{if } \omega_i \text{ is unknown} \end{cases} \tag{A.1}$$

    We opted for this one-hot-encoded version of (2.12) to maintain coherence with the code associated to [22].
- `r`
  - `data`: `.Rdata` file consisting of:
    - `e`: $N \times N_{\mathcal{L}}$ matrix defined according to (3.6);
    - `a`: $N \times N_{\mathcal{L}}$ matrix defined according to (3.7);
    - `c`: $N \times N_{\mathcal{L}}$ matrix defined according to (3.15);

  - ∗ p: $N \times N_{\mathcal{L}}$ matrix defined according to (3.8);
  - ∗ y: $N \times 2$ matrix defined according to (A.1).

- – geo: four files (.shp, .shx, .prj and .dbf) containing the set of polygons representing the roads. This will only be used for the visual results, so it may not be necessary if the algorithms are used for other GoOs, unrelated to the world of road pavement detection.

A couple of final specifications are necessary. For the code to work correctly, it is important that data points are stored in the same order in all data structures. Furthermore, the first $N_{\mathcal{L}}$ lines are supposed to refer to the labeled data points, while the following $N - N_{\mathcal{L}}$ lines have to be dedicated to the unlabeled ones.

Split

The first code available in the repository is split.R. Its role is to produce a train-test split assigning 80% of the data points to a training set and the remaining 20% to the test set. The user can change the split simply changing the seed at the beginning of the code.

The split is then transferred both to the python folder as a couple of .txt files, called train_mask and test_mask, and to the data.Rdata file in the r folder. In this way, we ensure that the split is the same for both algorithms, even if the language used is different.

κ-NN

In the knn folder, four different .R files can be found:

- knn_obj_cv: this code refers to Subsection 3.3.3. It performs cross-validation among all possible hyperparameters and then computes confusion matrix and pavement error over the test set.
- knn_obj_test: this code refers to Subsection 3.3.3 too. It computes confusion matrix and pavement error over the test set using the best hyperparameters.
- knn_goo_cv: this code refers to Subsection 3.3.4. It performs cross-validation among all possible hyperparameters and then computes confusion matrix and pavement error over the test set.
- knn_goo_test: this code refers to Subsection 3.3.4 too. It computes confusion matrix and pavement error over the test set using the best hyperparameters.

The user can also modify the dissimilarity matrix for the GoO-Oriented model by simply changing a line of code. In order for the most wide choice to be available, `p` and `a` have been added to `data.Rdata` even if we did not choose to use them in our final model.

GCN

The `gcn` folder contains everything necessary for the implementation of our Graph Convolutional Networks. This part of the code was written in Python, unlike the rest, which is in R, because the `tensorflow` package was required to train neural networks. Concerning this, all the required Python packages are listed in the `requirements.txt` file. Since version 1.15 of `tensorflow` was used, it is necessary for this code to be executed on Python 3.7 or an earlier version.

Together with some supporting code, mainly taken from [22] and then modified to address our situation, the main `.py` files to reproduce our results are:

- `gcn_edge_cv`: this code refers to Subsection 4.3.3. It performs cross-validation among all possible hyperparameters and then computes confusion matrix and pavement error over the test set.
- `gcn_edge_test`: this code refers to Subsection 4.3.3 too. It computes confusion matrix and pavement error over the test set using the best hyperparameters.
- `gcn_goo1_cv`: this code refers to Subsection 4.3.4. It performs cross-validation among all possible hyperparameters and then computes confusion matrix and pavement error over the test set.
- `gcn_goo1_test`: this code refers to Subsection 4.3.4 too. It computes confusion matrix and pavement error over the test set using the best hyperparameters.
- `gcn_goo2_cv`: this code refers to Subsection 4.3.5. It performs cross-validation among all possible hyperparameters and then computes confusion matrix and pavement error over the test set.
- `gcn_goo2_test`: this code refers to Subsection 4.3.5 too. It computes confusion matrix and pavement error over the test set using the best hyperparameters.

Plot

The final available file is `plot.R`. This is not strictly necessary for the classification itself, but it can be useful for the visual inspection of the results. It produces the plot of the results achieved by the algorithm that was ran last. The output is similar to the ones that we presented throughout the work: a map of the area of interest where roads are colored according to their predicted class. Paved roads are blue, unpaved roads are red and uncertain ones are purple.

Bibliography

[1] Sustainable Development Goals, 2015. URL https://sdgs.un.org/goals.

[2] Safari Njema, 2018. URL https://www.safari-njema.polimi.it.

[3] Google Earth, 2023. URL https://earth.google.com.

[4] OpenStreetMap, 2023. URL https://openstreetmap.org.

[5] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7(85):2399–2434, 2006.

[6] F. Berardo de Sousa and L. Zhao. Evaluating and comparing the igraph community detection algorithms. In *2014 Brazilian Conference on Intelligent Systems*, pages 408–413, 2014.

[7] N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936*. Clarendon Press, 1986.

[8] C. M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer New York, NY, 1 edition, 2006.

[9] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, Oct. 2008.

[10] A. Burzacchi, M. Landrò, and S. Vantini. Object-oriented classification of road pavement type in Greater Maputo from satellite images. *MOX Report*, 38, 2022.

[11] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 79(6):066111, 2004.

[12] G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[13] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[14] J. S. M. Dryden, Ian L. *Object Oriented Data Analysis*. Chapman and Hall/CRC, 2021.

[15] L. Duran-Lopez, J. P. Dominguez-Morales, A. F. Conde-Martin, S. Vicente-Diaz, and A. Linares-Barranco. Prometeo: A cnn-based computer-aided diagnosis system for wsi prostate cancer detection. *IEEE Access*, 8:128613–128628, 2020.

[16] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[17] E. Fix and J. Hodges. *Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties*. USAF School of Aviation Medicine, 1951.

[18] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3–5):75–174, Feb. 2010.

[19] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[20] J.-M. Guo, H. Markoni, and J.-D. Lee. Barnet: Boundary aware refinement network for crack detection. *IEEE Transactions on Intelligent Transportation Systems*, 23(7): 7343–7358, 2022.

[21] K. M. Gwilliam, V. Foster, R. S. Archondo-Callao, C. M. Briceno-Garmendia, A. Nogales, and K. Sethi. Africa infrastructure country diagnostic: roads in Sub-Saharan Africa. 2008. URL https://api.semanticscholar.org/CorpusID:44158882.

[22] T. N. Kipf and M. Welling. Semi-supervised classification with Graph Convolutional Networks, 2017. URL https://arxiv.org/abs/1609.02907.

[23] K. Mehlhorn and P. Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer, 2008.

[24] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119. Curran Associates Inc., 2013.

[25] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2018.

[26] National Imagery and Mapping Agency. Department of defense World Geodetic

System 1984: its definition and relationships with local geodetic systems. Technical Report TR8350.2, National Imagery and Mapping Agency, Jan. 2000.

[27] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6), 2004.

[28] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2), 2004.

[29] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[30] E. Pebesma. Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, 10(1):439–446, 2018.

[31] P. Pons and M. Latapy. Computing communities in large networks using random walks (long version), 2005. URL https://arxiv.org/abs/physics/0512106.

[32] QGIS Development Team. *QGIS Geographic Information System*. QGIS Association, 2023. URL https://qgis.org.

[33] A. Riid, R. Lõuk, R. Pihlak, A. Tepljakov, and K. Vassiljeva. Pavement distress detection with deep learning using the orthoframes acquired by a mobile mapping system. *Applied Sciences*, 9(22), 2019.

[34] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4): 1118–1123, Jan. 2008.

[35] M. Seeger. Learning with labeled and unlabeled data. 2000. URL http://infoscience.epfl.ch/record/161327.

[36] Z. Sun and K. Jia. Road surface condition classification based on color and texture information. In *2013 Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 137–140, 2013.

[37] G. J. Székely and M. L. Rizzo. Testing for equal distributions in high dimension. *InterStat*, 5:1249–1272, 2004.

[38] Z. Yang, W. W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings, 2016. URL https://arxiv.org/abs/1603.08861.

[39] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 912–919. AAAI Press, 2003.

List of Figures

List of Tables

List of Symbols

Graph of Objects Representation

Variable	Description
Ω	Set of all Objects
$\Omega_{\mathcal{L}}$	Labeled Objects in Ω
$\Omega_{\mathcal{P}}$	Paved Objects in Ω
$\Omega_{\mathcal{U}}$	Unpaved Objects in Ω
H	Set of all Edges
$\Gamma(\Omega, \mathrm{H})$	Graph of Objects

Matrix Representation

Variable	Dimension	Description
$\underline{y}$	N	True Class
$\underline{\hat{y}}$	N	Predicted Class
A	$N \times N$	Adjacency
E	$N \times N_{\mathcal{L}}$	Energy
C	$N \times N_{\mathcal{L}}$	Community
P	$N \times N_{\mathcal{L}}$	Shortest Path
Δ_A	$N \times N_{\mathcal{L}}$	Adjacency Dissimilarity
Δ_E	$N \times N_{\mathcal{L}}$	Energy Dissimilarity
Δ_C	$N \times N_{\mathcal{L}}$	Community Dissimilarity
Δ_P	$N \times N_{\mathcal{L}}$	Shortest Path Dissimilarity
Δ	$N \times N_{\mathcal{L}}$	Combined Dissimilarity

Ringraziamenti

La fine di questa tesi rappresenta anche la conclusione del mio percorso univeristario, dei miei cinque anni di studio per diventare ingegnere matematico. Mi sembra quindi più che giusto dedicarla a tutte le persone che mi sono state accanto durante il cammino.

In primis, un grazie va a tutta la mia famiglia, il mio più grande supporto in tutto quello che faccio, da sempre. Grazie ai miei genitori, per avermi dimostrato ogni giorno l'interesse più totale verso qualunque cosa io facessi. Grazie a mia nonna, per l'affetto incodizionato che solo lei sa dimostrare. Ma soprattutto grazie alla mia sorellina, che è cresciuta con me, contribuendo in gran parte a fare di me la persona che sono.

Grazie poi alla Maddi, che mi ha sempre fatto sentire speciale e al posto giusto. Non ci sono parole per esprimere quanto tu sia fondamentale. Lo so quanto difficile sia starmi vicino quando sono stressato per lo studio e questa tesi non è certamente stata da meno, grazie anche per questo.

Grazie ai miei amici, tutti quanti. Grazie agli impermeabili a ogni richiamo, Save e Mabe, che sono con me da più di un decennio e che ormai chiamare fratelli è praticamente scontato, e grazie ai Power Cringers, Giulio, Maci, Paparo e Jaja, siete stati i miei compagni di viaggio da quando sono arrivato a Milano e senza di voi non sarei qui. Grazie poi alla Eli, che spesso è stata l'unica compagnia in lunghe giornate di scrittura, e a Teo e Puri, per aver computato la maggior parte dell'immane matrice energetica. Grazie anche ai ragazzi della residenza Newton, sono anni che non vivo più con voi, ma una parte di questo traguardo è sicuramente vostra. Grazie a tutti gli altri, non ho spazio per nominarvi tutti, ma prometto che vi metterò nel sequel.

Grazie ovviamente al Professor Vantini, che mi ha assegnato questo lavoro e che spero non se ne sia pentito, e grazie ad Arianna, che ha scritto la tesi da cui è cominciato tutto e che mi ha seguito durante tutta la stesura. Lavorare con voi è stato un vero piacere.

Infine, grazie a te, se sei rimasto con Marco fin proprio alla fine.