Graph Convolutional Neural Networks for Open Data Discovery

- Bachelor's Degree Thesis -

Zoé Marie Meini

June 2021

Advisor: Sergi Nadal - Departament d'Enginyeria de Serveis i Sistemes d'Informació (ESSI)

Faculties: Facultat d'Informàtica de Barcelona (FIB)

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona (ETSETB)

Facultat de Matemàtiques i Estadística (FME)

Universitat Politècnica de Catalunya

BarcelonaTech

Acknowledgements

First of all, I would like to express my sincere gratitude to my advisor Sergi Nadal, who has guided and helped me throughout the whole realization of this thesis. He has been patient, present and offered me great encouragement whenever I needed it. I also appreciate that he trusted me to carry out a part of the *"An Automatic Data Discovery Approach to Enhance Barcelona's Data Ecosystem"* project because thanks to this opportunity I have been able to expand my knowledge in many ways.

I would also like to acknowledge professor Oscar Romero who pointed me towards this project when I asked him about any suggestions for my bachelor's degree thesis. He has also been present during the development of the thesis coming to several meetings and giving me very good advise.

I would like to greet Javier Flores who helped me with the NextiaJD tool very kindly when I required his assistance.

I also wish to thank Mariona González and Yasmina Assantuh, two of my dearest friends who helped me with their thoughtful counsels.

I could not have done this thesis without the help and the cheering of Roger Creus who has been my biggest support throughout this experience. I am grateful that I had him by my side.

This bachelor's degree has not been an easy road, but I feel very lucky that I met wonderful friends that I hope to keep for the rest of my life. Júlia Sala, Sara Sánchez, Roger Creus, Xavier Rubiés and Dànae Canillas, you have made the path worth the adversity, enjoyable and memorable.

Finally, I would like to thank my family who has given me unconditional support and never doubted my capacities to succeed in all the projects I engaged during the degree.

Abstract

In this thesis we study the suitability of a graph-based approach for data discovery in the context of Open Data BCN with respect to other traditional Machine Learning (ML) or Deep Learning (DL) methods such as Random Forest or Multi-Layer Perceptron. Data discovery consists on finding semantic relationships between datasets and their attributes (i.e., similar topic, joinable, unionable, etc.) in a massive open data repository. We tackle the dataset similarity task, which we reduce to a classification problem by defining two similar datasets as datasets sharing the same thematic area which we call: class. Exact and hash-based approaches have shown to be ineffective in such large environments, this is why we approach the problem from a learning perspective based on data profiles. We define profiles as representations that capture the underlying characteristics of the schemata and data values of datasets. Relying on profiles diminishes the complexity and enhances the model performances. To this goal, we provide both an efficient tool to extract datasets from the council's open data website and a Graph Neural Network (GNN) framework leveraging a graph representation of the Barcelona's repository by representing the datasets as nodes, the node embeddings as profiles and the edges as indicators of two nodes sharing a class. Specifically, we propose to solve a link prediction task using a Graph Convolutional Neural Network (GCNN) and to later map it into a classification task for each of the nodes in the graph. This mapping step is done with a novel function we designed called *Class_Selector* that uses both the amount of edges involving each of the classes and their corresponding probabilities. Our experiments show that both the baseline methodologies and our GCNN approach are able to learn from the training sets but have difficulties generalizing in the test phase due to the lack of data available in the repository. With respect to the baseline models, our approach obtains lower accuracy results but those could be improved by introducing *inductive link prediction* emerging solutions which are still under development.

Resum

En aquesta tesi estudiem la validesa d'un model basat en grafs per al descobriment de dades en el context d'Open Data BCN, respecte d'altres mètodes tradicionals d'aprenentatdge automàtic o profund com ara el Perceptró Multicapa o els Boscos Aleatoris. El descobriment de dades, conegut com a *data discovery*, consisteix en trobar relacions semàntiques entre conjunts de dades i els seus atributs en un repositori massiu de dades obertes (com per exemple, temes similars, punts d'unió, punts de "join", etc.). Abordem la tasca de semblança, que reduïm a un problema de classificació definint dos conjunts de dades similars com a conjunts de dades que comparteixen la mateixa àra temàtica que anomenem classe. Els enfocaments basats en comparacions de valors exactes o en funcions de hash resulten ser ineficaços en grans entorns de dades, per això abordem el problema des d'una perspectiva d'aprenentatge basada en perfils de dades. Aquests són representacions que capturen les característiques subjacents dels esquemes i els valors dels conjunts de dades. Basar-se en els perfils disminueix la complexitat i millora el rendiment dels models. Amb aquest objectiu, proporcionem una eina eficient per extreure conjunts de dades del portal de dades obertes de l'ajuntament i una implementació d'una Xarxa Neuronal basada en Grafs (GNN) que es beneficia de representar el repositori de Barcelona en forma de graf. Això s'aconsegueix codificant els conjunts de dades com a nodes, els seus perfils com a informació continguda al node (*embedding*) i un indicador de si dos conjunts de dades comparteixen classe com a una aresta. Concretament, proposem resoldre una tasca de predicció d'arestes mitjançant una Xarxa Neuronal Convolucional basada en grafs (GCNN). Posteriorment ho transformem a una tasca de classificació per a cadascun dels nodes del graf. Aquesta transformació es realitza amb una nova funció que hem dissenyat anomenada *Class_Selector* que utilitza tant la quantitat d'arestes relacionant nodes pertanyents a una classe específica, com les probabilitats assignades pel model a aquestes arestes. Els nostres experiments mostren que tant les metodologies bàsiques com el nostre model de GCNN són capaços d'aprendre dels conjunts de'entrenament, però que tenen dificultats per generalitzar en la fase de prova a causa de la manca de dades disponibles al repositori. Pel que fa al rendiment del nostre model, no s'aconsegueixen sobrepassar els resultats dels models més tradicionals. No obstant, el model de GCNN es podria millorar introduint solucions recents de *predicció d'arestes inductiuves* que encara estan en ple desenvolupament.

Resumen

En esta tesis estudiamos la validez de un model basado en grafos para el descubrimiento de datos en el contexto de Open Data BCN con respecto a otros métodos tradicionales de aprendizaje automático o profundo como los Bosques aleatorios o el Perceptrón multicapa. El descubrimiento de datos, conocido como *data discovery*, consiste en encontrar relaciones semánticas entre los conjuntos de datos y sus atributos en un repositorio masivo de datos abiertos (por ejemplo, temas similares, puntos de unión, puntos de "join", etc.). Abordamos la tarea de similaridad, la cual reducimos a un problema de clasificación al definir dos conjuntos de datos similares como conjuntos de datos que comparten la misma área temática a las que llamamos clase. Los enfoques basados en valores exactos o en funciones de hash se han mostrado ineficaces en entornos tan grandes, por eso abordamos el problema desde una perspectiva de aprendizaje basada en perfiles de datos. Estos se definen como representaciones que capturan las características subyacentes de los esquemas y valores de los conjuntos de datos. Utilizar los perfiles disminuye la complejidad y mejora el rendimiento del modelo. Con este objetivo, proporcionamos una herramienta eficiente para extraer conjuntos de datos del repositorio de datos abiertos del ayuntamiento de barcelona y una implementación de una Red neuronal basada en grafos (GNN) que se beneficia de la representación del repositorio de Barcelona como un grafo. Esto se consigue al codificar los conjuntos de datos como nodos, sus perfiles como información contenida en el nodo (*embedding*) y un indicador de dos nodos compartiendo clase como una arista. Específicamente, proponemos resolver una tarea de predicción de aristas utilizando una Red neuronal convolucional basada en grafos (GCNN). Esta tarea es posteriormente transformada a una tarea de clasificación para cada uno de los nodos del grafo. Esta transformación se realiza con una nueva función diseñada a la que llamamos *Class_Selector*. Esta usa tanto la cantidad de aristas que involucran a cada una de las clases, como las probabilidades correspondientes que el modelo les ha asignado. Nuestros experimentos muestran que tanto las metodologías más tradicionales como nuestro modelo de GCNN pueden aprender de los conjuntos de entrenamiento, pero tienen dificultades para generalizar en la fase de prueba debido a la falta de datos disponibles en el repositorio. Nuestro modelo de GCNN no consigue alcanzar los resultados obtenidos por las propuestas más tradicionales. No obstante, estos resultados podrían mejorar mediante la introducción de soluciones emergentes de *predicción de enlaces inductivos* que están actualmente en pleno desarrollo.

Contents

1 Introduction

1.1 Motivation

For years now, data science has been defined as *"The sexiest job of the 21st century"* [1]. But being a data scientist involves facing a vast number of challenges before arriving at the specific tasks that are considered to be the heart of their job. At the very beginning they are limited by, not only the quantity and quality of data, but the capacity of evaluating and extracting useful information from it.

Nowadays, the hardest task to solve and the one requiring most time, consuming approximately an 80% of the time spent in a data science project, is data integration, which is mainly composed by data cleaning and data discovery. Moreover, 76% of data scientists also consider those parts as the worst of their jobs [2]. Not only it is time-consuming, but it is highly exhausting as it gets aggravated by the propagation of large-scale repositories, such as open data repositories or data lakes, which host large amounts of unstructured and heterogeneous data, making it hard to obtain useful insights from it without a previous data preparation process. [3, 4].

Automating these kind of tasks in such complex environments by taking advantage of the power offered by emerging techniques such as Artificial Intelligence (AI) can be a good opportunity to ease, fasten and improve those arduous processes. This would give the benefit of spending more time in focusing on solving specific data science problems and offer better solutions at the end.

This thesis focuses on profit from arising Deep Learning (DL) models called Graph Neural Networks (GNNs) to help carrying out data discovery tasks proposing self-regulating methodologies. The asset of using those models relies on the ability they have to exploit further aspects than the ones we could exploit using other Machine Learning (ML) or DL approaches, such as connectivity and topology. As an added value, in the data discovery context GNNs have not yet been delved into as they are still a very novel AI-research topic that requires further exploration for fully being aware of their rich potential.

The project is specifically centred in Open Data Discovery for the city of Barcelona, as it takes part of a bigger project for the city council: *An Automatic Data Discovery Approach to Enhance Barcelona's Data Ecosystem* carried out by the *Database Technologies and Information Management Group* (DTIM), a research group in *Universitat Politècnica de Catalunya* (UPC).

1.2 Data Discovery

We define as data discovery (also called dataset discovery by some authors[5]) the process of navigating numerous data sources in order to find relevant datasets as well as the relationships among those. Data discovery takes part of data integration which along data cleaning constitute the most tedious, cumbersome and confusing data science tasks. Moreover, data integration is deemed to be one of the hard Computer Science unresolved problems as it encompasses volume, velocity, variety and veracity of data. Not only integrating known data hinders getting insights and making analysis, but to that is added a more substantial problem which is finding the right

data in a large and heterogeneous environment to solve a data science problem [6]. This is the heart of data discovery.

Data analytics is becoming increasingly valuable for business operations throughout all industries as extracting and analyzing KPIs or organization metrics allows the companies to gain insights into their capabilities to optimize their general performance. In business analytics data discovery is also defined as: *"The collection and analysis of data from various sources to gain insight from hidden patterns and trends. Through the data discovery process, data is gathered, combined, and analyzed in a sequence of steps. The goal is to make messy and scattered data clean, understandable, and user-friendly."* [7]. That is why we say that data discovery requires skills in understanding data relationships and data modeling as well as in using data analysis and guided advanced analytics functions as its ultimate goal is to boost a dataset with information previously unknown to the user [5].

The greater part of data discovery, focuses on tabular data, which is data structured into equally-dimensional rows containing the same attributes providing values to those rows. This is because it frames the main form of datasets in both enterprises and the web, often found under CSV formats, spreadsheets, database relations and web tables[1].

There are a lot of tasks constituting data discovery, but the main are:

- Finding similar datasets to a given one, defining ways to measure their similarity.
- Finding unionable and duplicates of datasets to augment them with additional attributes that furnish extra information.
- Finding joinable datasets to cross data to expand it with new an interesting related topics.

The tasks above are presented in decreasing complexity. In this thesis we will attack the very first one. Its difficulty relies on the definition or the decision of the similarity metric used to compare datasets. Sometimes similarity between datasets can be determined by its labels, if they are specified, or by their category, if they belong to a class of any kind, but other times it is very hard to establish a measure that is able to capture the resemblance between tables. The second and third ones, finding unions, duplicates and joins go a step further in terms of difficulty as they demand special treatments to capture the right associations between chunks of data. Something that can hinder the complexity of the tasks is changing their granularity. Indeed, we can perform data discovery at two different levels: we can do it at *dataset level* or at *attribute level*. Both of those modalities have a handful techniques and approaches. The most common are:

- Comparison by value: approach that compares exact values. It is characterised by the use of auxiliary data structures, which are helper data structures to solve a given problem, such as dictionaries, lists, inverted indexes, queues, and more, to minimize the lookup cost.

[1] In the context of this thesis we will work only with CSV formats as it is the only format in which tabular data is provided by Open Data BCN repository.

- Comparison by hash: approach that relies on the fact that applications frequently read or write data that is identical to already existing data [8]. It uses hash functions to map variable length inputs to fixed length outputs and then compare them to see if they are similar. This technique also employs index structures for efficient threshold index.

- Comparison by profile: based on the meta-feature extraction of datasets (or attributes) to form profile tables called profiles, that will be compared to apprise similarities or shared information between them.

Search accuracy		
Exact. . . . . .Approximate		
Comp. by value [9] [10] [11]	Comp. by hash [12] [13] [14] [15]	Comp. by profile [16] [17] [18]
Expensive. . . . . .Efficient		
Algorithimc complexity		

Table 1: Overview of approaches by technique. These are further arranged according to the kind of search and their algorithmic complexity. The table is extracted from [19].

The first two approaches have some disadvantages that the last one does not have. One of them is that in big data the use of index structures demands a high amount of computing resources so that we can maintain them properly. The other drawback is that given that the similarity measure employed is either containment[2] or Jaccard distance[3], precision is highly affected since they produce a large number of false positives [19].

Contrarily, methodologies based on profiles comparison remain unaffected by the liabilities mentioned above. Moreover, not using data values directly but a summary formed by their meta-features shrinks the amount of data to deal with, diminishing the complexity of the problem and in consequence improving the efficiency of the solution. Due to all of those benefits, this approach is highly used when wanting to introduce ML in data discovery[19]. Indeed, using profiles enhances the use of models that can learn from them to predict similarities, unions or joins. Although all of these benefits appear to be appealing, there also are some drawbacks when using profile-based methods. Experimentally we can observe a large gap in the trade-off regarding the quality of the results, in terms of predictive performance, obtained with this approach compared to hash or value comparison methods. Which is why we aim to explore advanced techniques that take benefit of the natural graph-structure and the interrelationships between datasets and attributes which could help on finding new ways to improve profile-based approaches by keeping its scalability advantage and trying to improve its predictive performance.

[2]The containment similarity measures similarity between finite sample sets, and is defined as the size of the intersection of the sets divided by the size of one of them: $containment\_similarity(A, B) = \frac{|A \cap B|}{|A|}$

[3]The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets: $Jaccard\_distance(A, B) = \frac{|A \cap B|}{|A \cup B|}$

Overall, in data lake settings[4], profile-based methods are clearly the way to go [20]. Those methods also carry some challenges such as how to create the profiles of the datasets, or their attributes, in a way that they are easy to compute and compare while still being representative enough to ensure a good learning approach on them to obtain scores of accuracy and precision that could be comparable to the ones obtained with non-scalable techniques.

1.3 Open Data Discovery

1.3.1 Open Data

Open data is defined as data that is publicly accessible for all to use, republish and build-on, with no limitations imposed by copyright or any control mechanisms. This description is based on three key concepts which explain why Open Data is so strong: i) Availability and Access, ii) Re-use and Redistribution, and finally iii) Universal Participation [21].

While the principles underlying open data exists and has been well-established for a long time, the concept of *open data* is relatively new, having gained traction with the advent of the internet and World Wide Web, as well as the introduction of open-data government initiatives. In fact, Open Government Data (OGD) is probably one of the most important forms of open data. It is spreading in governments all over the world because it helps making their data more complete, primary, and timely [22]. It is a concept that encourages openness, accountability, and value development by rendering government data accessible to everyone. Large amounts of data and documents are generated and commissioned by public institutions. By making their datasets available those institutions become more transparent and accountable to citizens.

Many countries have add up to this initiative by opening portals of open data managed by their city councils. Examples of that are: Canadian Open Portal[5], Berlin Open Data[6], Data GO JP[7], Open Data BCN[8], Paris Data[9], and others.

Something we can notice about those kind of portals is that:

- Their datasets are always categorized in similar topics forming classes of labeled datasets.
- The datasets contained in each of the class may follow the same schema: same columns and dimensions.
- The structure of a given dataset is very provider-dependant, it changes depending on from where or whom it was obtained, creating an heterogeneous repository.
- Data is denormalized, meaning that it does not implicitly contain relations among tables using primary and foreign keys. They have to be related explicitly.

[4] set of centralized repositories containing vast amounts of raw data (either structured or unstructured), described by metadata, organized into identifiable data sets, and available on demand

[5] `https://open.canada.ca/en/open-data`

[6] `https://daten.berlin.de/`

[7] `https://www.data.go.jp/?lang=en`

[8] `https://opendata-ajuntament.barcelona.cat/es/`

[9] `https://opendata.paris.fr/pages/home/`

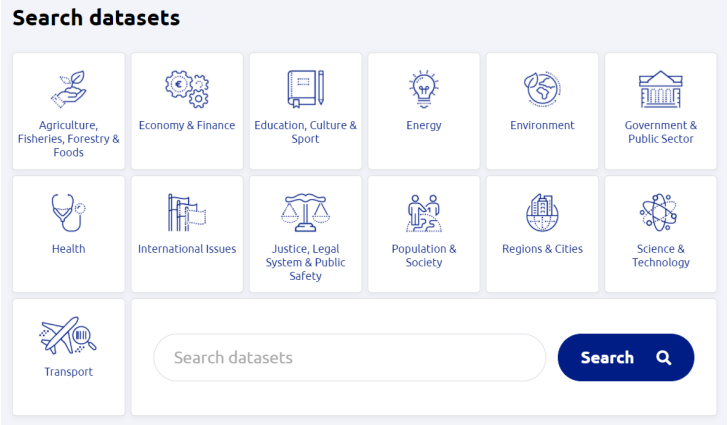


Figure 1: Example of data distribution of the Data Europa EU portal

Although this is an increasing movement, for now, open data remains an exception. Encouraging open data initiatives, a foundation called Open Data Barometer[10] measures how governments are publishing and using open data for accountability, innovation and social impact. It aims to uncover the true prevalence and impact of open data initiatives around the world by measuring readiness, implementation and impact of the data stated as "open" by the countries.

1.3.2 Data Discovery in Open Data

Initiatives like open data encourage the shareability of data coming from many different sources and in consequence we notice a growth in its diversity, generating repositories of heterogeneous data. We may think of variety as a positive aspect because it allows an increase in the amount of knowledge and information in our data, but if open repositories are not maintained and their data is not well integrated they can become very clutter and turn into data swamps[11] loosing all its interest. Efforts to support genuine openness often include specialized teams who manually collect or analyse data from tens or hundreds of sources and laboriously edit, assemble, classify and label it, among others, to generate different categories or topics.

Taking into account the importance of open data, particularly OGDs, it is key to provide tools to handle properly the data integration step. This is where data discovery plays a very important role and why it is crucial to automate it by introducing AI rather than letting those tasks be solved manually. Scientists are aware of this, and hence, there are many initiatives working on data discovery for open data portals/repositories. For example, in [22] Renée J. Miller et al. propose different approaches to perform data discovery, specifically finding joinable and unionable datasets, on open data. They experiment with the Canadian Open Portal and present ideas based on LSH, search engines and connectivity graphs. Nonetheless, it leaves some problems open, remarking that data discovery is not an easy task. At the end, they give table join and table union search solutions that provide interactive search speed over massive collections of

[10] `https://opendatabarometer.org/?_year=2017&indicator=ODB`

[11] A data swamp is an unstructured, ungoverned, and out of control data lake where due to a lack of process, standards and governance, data is hard to find, hard to use and is consumed out of context.

attributes with heavily skewed cardinality distributions. Through examples, they also highlight the power of search solutions making open data more transparent and usable.

Nevertheless, one of the drawbacks of using LSH methods is that it is not a scalable approach and, for growing repositories such as open portals, this can become a limited solution. We state, that for data discovery in open data the most suitable approach would be the one based on profiles comparisons.

1.4 Use case: Open Data Discovery in Barcelona

1.4.1 Open Data BCN and Barcelona's Data Ecosystem

The city of Barcelona has its own open data portal called Open Data BCN. Open Data BCN was born in 2010 and is managed by the Barcelona's city council, more specifically the Department of Statistics and Data Dissemination of the Municipal Data Office. This platform has as goal to provide open and accurate data related to Barcelona's citizens and infrastructures. It currently has approximately 490 datasets categorized in different thematic areas as shown in Figure 2.

The amount of datasets in each category is not balanced, being *city and services* the one with most datasets (around 160) and *economy and business* the one with less data (around 20 datasets). Exploring the classes and the type of data contained in each, we can see that for some cases we have multiple instances of a same dataset but for different years, type of vehicles, buildings, etc., which makes us think that these have been generated slicing a multi-dimensional dataset, and in consequence they share the same attribute structure.

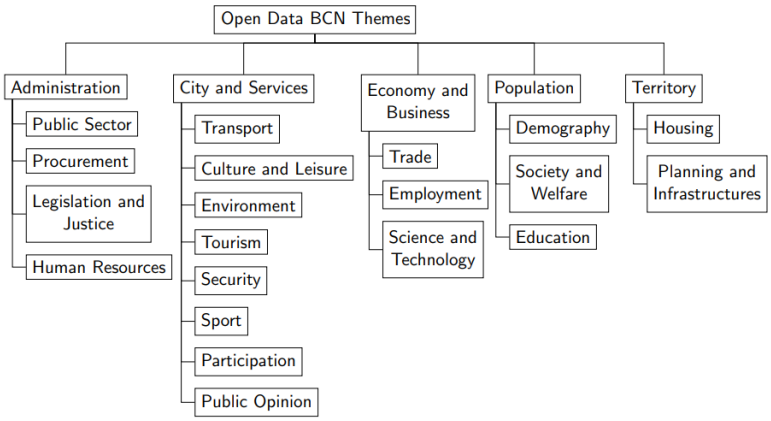


Figure 2: Thematic areas in the dataset catalogue of Open Data BCN

In Open Data BCN, data are displayed in many file formats such as: CSV, DGN, DWG, SHP, KMZ, WMS, JSON, WMTS, XML, RDF, among others, being the CSV the most frequent.

The Barcelona's Data Ecosystem is also formed by other open data portals such as: Tourism Data

System[12], Institut d'Estadística de Catalunya[13], Àrea Metropolitana de Catalunya[14], Real-time Bicing data[15], BIOCOMSC COVID-19 reports[16], Meteocat REST API[17], among others.

There also are data sources owned by third-party organizations that demand subscription mechanism in order to have access to their data. The type of data produced by these platforms is crucial for studying population and their behaviour. There are data providers that span the entire scope of data relating to Barcelona's urban challenges, such as: tourism, food supply, population, well-being, mobility and housing and basic services. These kind of data has harder access as organizations keep them behind pay-walls.

One of the limitations that the Barcelona's Ecosystem has is that it is somehow composed by independent sources of data making integration a non-trivial process. Those data sources become isolated silos with unlinked data generating a *vertical view*. Whereas addressing Barcelona's urban challenges intelligently would require having an *horizontal view*, which is achieved by finding, contextualizing and crossing data.

1.4.2 Problem Formalization

This thesis focuses on solving the similarity task of data discovery, using tabular data (CSV format), specifically in the Open Data BCN context. To formalize our problem we define the following concepts:

Profiles. A profile P for a dataset D_n, reffered as $P(D_n)$ is a set of both numeric and nominal meta-features $m_1, ..., m_m$. Each m_i is a statistic about the content of an attribute of D_n.

Similarity. Let $A = \{A_1, \ldots, A_k\}$ be a set of thematic areas, and $D = \{D_1, \ldots, D_m\}$ a set of datasets. Each D_i belongs to a thematic area A_i (denoted $t(D_n) = A_i$). We say two datasets D_i, D_j are similar if $t(D_i) = t(D_j)$.

Similarity function. Let $D = \{D_1, \ldots, D_m\}$ be an open repository composed by tabular datasets. We define a function $F(P(D_i), P(D_j))$, such that for a pair of profiles $P(D_i)$, $P(D_j)$ it outputs `true` if $t(D_i) = t(D_j)$ or `false` otherwise.

In other words, two datasets share the same thematic area, and hence, are similar. The problem of finding similarities among datasets will be reduced to predict their thematic area to see whether they belong to the same or not. Hence, given a new unlabeled dataset what we want is to predict its thematic area to know its similarity with other datasets of the repository.

The rest of the thesis is devoted to present our approach of the similarity function F using a GNN model to predict similarities among datasets of the Barcelona Open repository.

[12] `https://barcelona.tourism-data-system.cat/`
[13] `https://www.idescat.cat/emex/?id=080193`
[14] `https://www.amb.cat/s/web/area-metropolitana/dades-obertes.html`
[15] `https://www.bicing.barcelona/dades-bicing`
[16] `https://biocomsc.upc.edu/en/covid-19`
[17] `https://apidocs.meteocat.gencat.cat/`

1.5 Graph Neural Networks for Open Data Discovery in Barcelona

1.5.1 Background

In the last decade, neural network models have been developed successfully for structured data such as text, images, audios or videos. The connection structure between layers makes neural networks suitable for processing signals in tensor forms where the tensor elements are arranged in a meaningful order. This fixed input order is fundamental for neural networks to extract higher-level features [23]. Nevertheless, in certain cases data can be presented under complex relational forms that lack a tensor representation with fixed ordering. An example is graph-structured data. Graphs are universal data structures made up of nodes and edges that can represent complex relational data. Additional challenges arrive with these kind of structures making traditional deep learning limited to handle complex relational data. For example, convolutions, which are typical operations of networks, cannot be explicitly extended to irregular graph domains. These challenges, sparked the development of Geometric Deep Learning (GDL) research, which aims to apply deep learning techniques to non-Euclidean data and since graphs are so common in real-world implementations, there has been a surge of interest in applying learning methods to them [24].

In the past few years, Graph Neural Networks (GNNs) have emerged as an effective class of models for learning embeddings of nodes. GNNs are based on a recursive neighborhood aggregation (or message passing) scheme, in which each node aggregates the feature vectors of its neighbors to compute its own new feature vector [25]. These representation learning techniques learn a mapping that embeds nodes, or even entire graphs, as points in a low-dimensional vector space $\mathbb{R}^d$. The objective is to optimize this mapping such that geometric relationships in the embedding space match the original graph's structure. The learnt embeddings can be utilized as feature inputs for downstream ML tasks after the embedding space has been optimized (i.e. schema matching or entity alignment) [26].

1.5.2 Proposal and Contributions

Data integration tasks such as data discovery can be vastly enhanced by the use of arising deep learning models based on graphs. Indeed, when we need to go beyond attribute name comparison, graph data structures can help in catching data distributions and their relationships. This is possible thanks to the flexibility and interoperability of the graph data model, where instead of storing all instances of a dataset we can store succinct and informative summaries of them.

This is why, in order to efficiently and effectively address large-scale data discovery, we propose to introduce GNN models to solve profile-based approaches for the similarity task at dataset level. We would represent the different elements of the Open Data BCN repository using graphs, where the nodes would denote the datasets with arbitrary complex meta-features. We would call their data profile as the node embeddings, and the edges would denote similarities between the aforementioned datasets. Using the proposed graph representation of the input data, we can learn node embeddings, and perform matching operations in the embedded space. In order to perform comparisons for the matching tasks, we must obtain a convenient

representation that is able to capture both, structural and semantic information of the nodes in the graph. For all the reasons stated above, we aim to explore the Graph Convolutional Network (GCN) approach, to generate a vectorized representation of the nodes (embeddings) in the graph.

Particularly, we make hypotheses summarized in the following research questions that the thesis will address:

- ***RQ1: Can Graph Neural Network-based techniques be used for data discovery in Open Data BCN repository?***
- ***RQ2: Can we notice an improvement in the performance (in terms of accuracy) of GNN approaches in front of more classical data discovery methodologies?***

To approach these research questions we will work on both a GNN model and more traditional ML and DL techniques.

1.5.3 Novelty of the Proposal

In this proposal we advocate for a novel GNN-based approach for data discovery on large-scale repositories of heterogeneous, independently created datasets. Our work is motivated by (i) the poor predictive performance of current profile-based solutions, and (ii) the inability to scale-up of hash-based approaches, as well as their low precision, which is undesirable for large-scale scenarios. There are other works proposing to learn embeddings from graph representations [27, 28, 29, 30], or directly from the columns of tabular datasets. However, these leverage instance data from the datasets and will thus fail at large-scale data lakes such as Barcelona's. To the best of our knowledge, the proposal is unique as it combines computation of large volumes of data while retaining high predictive quality.

2 Preliminaries

2.1 Classification Predictive Modeling

In machine and deep learning, classification refers to a predictive modeling problem where a class label is predicted for a given example of input data. Commonly classification models estimate a continuous value as the probability of a given example belonging to each output class. The probabilities reflect the likelihood or confidence that a given example belongs to each class. By choosing the class label with the highest probability, an expected probability may be translated into a class value. In order to validate the classification capacity of the model there are many available metrics (i.e. accuracy, precision, recall, F-score).

There are different types of classification tasks:

- **Binary Classification**: It consists in assigning an element into a set of two classes using a classification rule or a classifier. Typically, binary classification tasks involve one class that is the normal state and another class that is the abnormal state. Commonly, binary classifiers are modeled by predicting a Bernoulli probability distribution for each data instance. The higher outcome of probability of each class will be the one assigned to the element. For example, classifying whether a new is *real* or *fake* is a Binary Classification problem.
- **Multi-Class Classification**: It generalizes Binary Classification to problems where there exist multiple options of class membership. In these problems, the output is a discrete probability distribution over the set of possible classes. As in Binary classification, we consider a unique class membership for each observation. For example, the classification of facial expressions from images can define a multi-class classification problem since a face could either be *sad*, *happy*, *angry*, *surprised* or other.
- **Multi-Label Classification**: Compared to classes, labels are not considered unique annotations for input observations. In this way, label classification can tag observations with different number of annotations. For instance, object detection from images is a multi-label classification problem since an image could contain only a dog, or could contain both a dog and a cat, or a dog, a cat and a person.

In machine learning, some algorithms that are typically used for solving classification problems are: *Logistic Regression* (*LR*) [*31*], *Decision Trees* (*DT*) [*32*], *Support Vector Machines* (*SVMs*) [*33*], *k-Nearest Neighbours* (*k-NN*) [*34*]. In these cases, an important part of the performance of the models depends on the observation representations. Depending on how the observations are encoded, these can belong to dimensional spaces where it can both be easier or more difficult to classify in. It is a common challenge in machine learning to perform accurate feature engineering so that the classification framework is suitable. Moreover, feature engineering can be complicated with tabular data but even more with more complex data structures like images or text. For this reason, machine learning extends to deep learning by using sophisticated artificial neural network architectures. Neural Networks are complex and robust architectures adaptable to a wide range of domains. The later is possible since Neural Networks carry an embedded rep-

resentation learning system that allows them to learn useful feature representations from raw data in an autonomous way. However, they demand larger quantities of data, which is not actually a severe problem nowadays. With all this, different configurations of Neural Networks have succeeded in solving classification problems (among many others) in domains as complex as images, videos and text.

2.2 Graph Neural Networks (GNNs)

With the usage of deep learning architectures the AI community has solved problems in domains that traditional ML could not tackle. Either image-based (i.e. image classification, object segmentation, video trajectory tracking), text-based (i.e. text semantic recognition, question answering) or voice-based (i.e. speech understanding or generation) problems cannot be approached with ML because they are not based on input tabular data. In this way, Neural Networks are well-suited to processing data in tensor forms, with the tensor components ordered in a meaningful order, thanks to the connection structure between layers. In fact, computer vision and Natural Language Processing are two fields of study that have experienced revolutionary results through DL (i.e. Recurrent Neural Networks (RNN) and Convolutional Neural Networks(CNN)). Even though images, text or audio signals propose more challenging setups than tabular data for autonomously learning patterns, in many settings data can have even more complex structures that lack a tensor representation with fixed ordering and can not be tackled with standard NNs. An example are graphs, which do not have sequential nor grid-like representations and do not provide an embedded spatial locality property or a fixed node ordering and hence can not be tackled with standard NNs. For those structures we use Graph Neural Network (GNN) approaches.

2.2.1 Graphs and Graph embeddings

Formally, we define a **graph** as a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = v_1, \ldots, v_{|\mathcal{V}|}$ is the *vertex or node set* and $\mathcal{E} = e_1, \ldots, e_{|\mathcal{E}|}$ is the *set of edges* that connects them, where each edge $e_k \in \mathcal{E}$ represents a pair of nodes (v_i, v_j) with $v_i, v_j \in \mathcal{V}$. We say a graph is **weighted** if there exists a function that assigns a weight to an edge between a pair of nodes: $w : (v_i, v_j) \rightarrow w_{ij}$. Contrarily, we say that the graph is **unweighted**. We define a **directed** graph as a graph in which the order of the node-pair (v_i, v_j) given by an edge e_k is not permutable, as (v_j, v_i) may not belong to $\mathcal{E}$. On the other hand, we say a graph is **undirected** if the fact that $(v_i, v_j) \in \mathcal{E}$ implies that $(v_j, v_i) \in \mathcal{E}$. Then for an edge e_k with (v_i, v_j) and an edge e_l with (v_j, v_i) we have that $e_k = e_l$ and that the relations between nodes are symmetric. The definitions are taken and adapted from [24]. We can represent a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with an **Adjacency Matrix** $A \in \mathcal{M}_{|\mathcal{V}|x|\mathcal{V}|}$ whose elements indicate whether pairs of vertices are *adjacent* (they define an edge) or not. We assume the matrix binary, with an arbitrary element from A, $a_{ij} = 1$ meaning that the pair $(v_i, v_j) \in \mathcal{E}$ and $a_{ij} = 0$ otherwise. We also define the **Degree Matrix** $D \in \mathcal{M}_{|\mathcal{V}|x|\mathcal{V}|}$ as a diagonal matrix containing the number of edges attached to a node. Hence, the count of edges containing v_i is stored in the i-th item of the diagonal. Additionally, we define the neighbourhood of an arbitrary node v_i, $\mathcal{N}(v_i)$ as the set of the nodes v_j such that there exists an edge $(v_j, v_i) \in \mathcal{E}$.

Recently in DL there has been an emerging interest in learning from graphs since there exist large amounts of data that are encoded in this format, such as social networks, biological structures like proteins and molecules, among others. That is why Graph Representation Learning (GRL) methods, which aim at dealing with these kind of data, have gained popularity in the last decade. As other typical NNs approaches, GRL learns from low-dimensional vector representations of the graphs called **graph embeddings** or **network embeddings** [27, 24]. The goal of converting graph attributes to vectors is to make graph data useful as inputs for DL models such as NNs, which have been shown to operate well with tensor data. Node embeddings encode nodes so that similarity in the embedding space approximates similarity in the graph.

Formally, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ represented with an adjacency matrix $A \in \mathcal{M}_{|\mathcal{V}|x|\mathcal{V}|}$, we define low-dimensional vector representations $\{Z_i\}_{i \in \mathcal{V}}$ for nodes $v_i \in \mathcal{V}$ in the graph called **graph embeddings**, such that graph properties are preserved [24]. For instance, if two nodes have similar connections in the original graph, their learned vector representations should be close. We denote as $Z \in \mathbb{R}^{|V| \times d}$ the node embedding matrix.

2.2.2 Neural Message Passing

The major common approach for tackling graphs with Neural Networks is with **message passing**, which tries to generalize the convolution operation present in classical DL approaches to the graph format. Message passing works over the key idea that the feature information of the node is passed through the structure of the graph and updated to the node that receives it. We define the states of the node v_i at a specific time t: h_t^i as the embedded information of this node at time t. Message passing has the following steps:

- Fetch the current $(t-1)$ neighbour states of a node v_i: $h_{t-1}^j, \forall v_j \in \mathcal{N}(v_i)$.
- Prepare the messages the node will receive from its neighbours: $f(h_{t-1}^j), \forall v_j \in \mathcal{N}(v_i)$.
- Summarize/Aggregate the information received by the node obtaining the new node state for v_i: h_t^i.

In the message passing framework, the step of information update happens at the same time for all the nodes of the network. The preparation and aggregation processes are shared among all nodes. Besides, the number of update steps to perform is a user-defined hyper-parameter. Another particularity of message passing is that what we call the **node receptive field** of a node, which is the set of nodes passing a message to a specific node, increases at each step with the node neighbourhood.

2.2.3 Graph Neural Networks (GNNs) and Graph Convolutional Neural Networks (GCNNs)

Any GNN can be understood over the idea of neural message passing. In the definition of a generic GNN we define h_0^i as the initial features of node v_i and h_T^i its embedding after T updates. Then the general update formula of a node is:

$$h_t^i = \sigma\left(Uh_{t-1}^i + W\sum_{v\in\mathcal{N}(i)} E_k h_{t-1}^j\right)$$

Where the matrices U, W, and E_k are the GNN trainable weights. U is learnt such that the important information of the node embedding in the previous update h_{t-1}^i is preserved. W is learnt such that the valuable information in the nodes $v_j \in \mathcal{N}(i)$ is accordingly passed to the node v_i ($\mathcal{N}(v_i)$ being the neighbourhood of node v_i). Then E_k defines the way the edge attributes influence the information passed from neighbouring nodes $v_j \in \mathcal{N}(v_i)$ to v_i. Finally, $\sigma(x)$ is a non-linear activation function that enables the GNN to estimate complex functions. With all this, we can feed the final embeddings of the nodes to a suitable loss function for the given task and update the GNN weights to minimize the loss (i.e. with Stochastic Gradient Descent).

A popular and more specific approach to define a GNN is the Graph Convolutional Neural Network (GCNN) defined as:

$$h_t^i = \sigma\left(\sum_{v_j\in\mathcal{N}(i)} \frac{W_t h_{t-1}^j}{deg(i)}\right) \tag{1}$$

Where we extend the convolution operation to graphs by learning a shared weight matrix W that effectively transforms and aggregates the information between neighbouring nodes. The matrix form of Equation 1 is the following.

$$H_t^i = \sigma\left(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}H_{t-1}W_t\right) \tag{2}$$

Where matrices A and D are the adjacency and degree matrices respectively and H_{t-1} stores the node embeddings of all the nodes in the previous update.

The matrices A and D are usually modified to include *self-loops*. These are edges that are added from each node to itself. In this way, we open a channel for nodes to maintain a degree of information from their own embedding in the last update.

2.2.4 Learning Tasks on Graphs (Applications)

The objective of GNN in a graph is to learn meaningful node embeddings that encode the node features in a meaningful way. Once we have done this, we can use that graph model for many different applications. In this section we provide a brief overview of the most important and well-studied tasks we can solve on graph data. We take the examples from [35].

- Node Classification:

In node classification, the goal is to predict the label or the class associated to the nodes $v \in \mathcal{V}$ when only the true labels of a specific set of nodes are given, which are used as the training

set. Node classification has become through the years the most popular ML task on graph data. Some examples are: classifying the function of proteins in the interactome [36] and classifying the topic of documents based on hyperlinks or citation graphs [37]. Node classification is useful for inferring information about a node based on its relationships with other nodes in the graph.

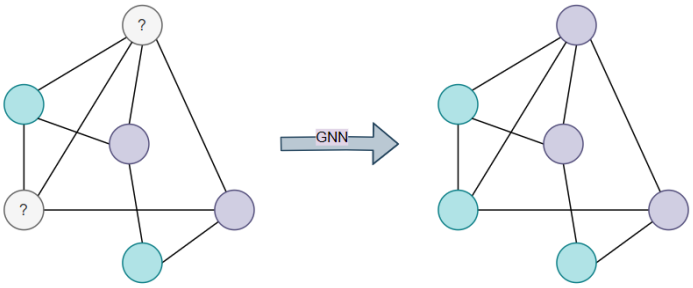


Figure 3: Example of node classification. The GNN classifies the nodes that did not belong to any class.

- Link Prediction:

Sometimes we are missing the information about relationships between nodes in the graph and we would need an approach to infer those edges. This is the task that link prediction does which besides node classification, is also one of the most popular ML tasks with graph data. Formally, given a set of nodes $\mathcal{V}$ and an incomplete set of edges between those nodes $\mathcal{E}_{train} \subset \mathcal{E}$, which is used as training set, the goal is to infer the missing edges $\mathcal{E} \setminus \mathcal{E}_{train}$. The approach can also be modified to remove edges by setting a threshold on the probability score given to an edge. Some examples are: recommending content to users in social platforms [38] or inferring new facts in a relational database [39]. The complexity of this task is dependant on the type of graph data that we are dealing with. If, for instance, the graphs have multi-labeled edges, the difficulty of detecting new edges on this graph increases.

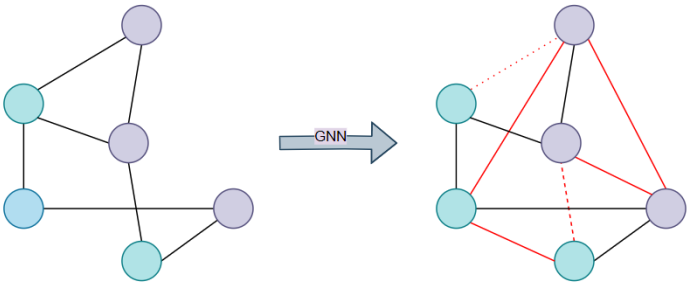


Figure 4: Example of link prediction. The GNN finds the new edges in red and removes the edges represented in dotted lines.

- Graph Classification:

In graph classification the goal is to make predictions on the type or class of each of the graphs within a multiple graph dataset. Given a set of graphs $G = \{g_1, g_2, \ldots, g_n\}$ and a set of classes

$C = \{c_1, c_2, \ldots, c_m\}$ we want to assign a class c_i to each graph of G. See as examples: a regression model that can predict whether a molecule is toxic or soluble [40], or a classification model to detect if a computer program is malicious [41].

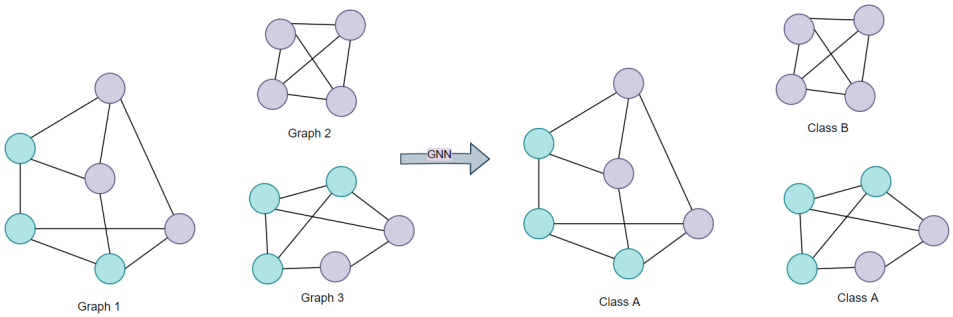


Figure 5: Example of graph classification. The GNN assigns a class to each of the graphs

- Community Detection:

In a graph, a community is defined when nodes are linked between them more densely than in other parts of the graph. The objective of community detection is given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, to infer its existing community structures. For example, uncovering fraudulent groups of users in financial transaction networks [42].

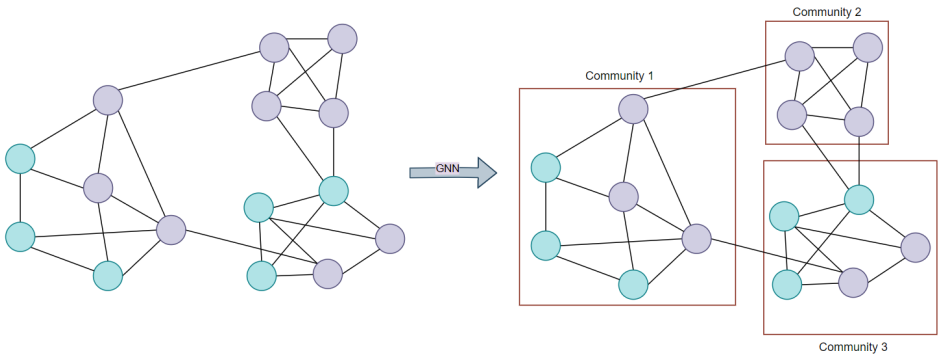


Figure 6: Example of community detection. The GNN finds communities within a graph.

3 Data Collection and Preparation

Any data-related journey needs to start with the acquisition and preparation of the data to be analyzed. Before starting, we need to ask ourselves questions such as, *"where do I collect the data from?"*, *"what kind of data do I need"* and *"what kind of collection tools or channels are available?"*. There are many types of data collection, for example: surveys, records, clinical studies or forms. Data can be obtained either from a primary source, (collected by yourself through various means) or a secondary source (government data sets or online repositories).

In this thesis, the focus is on solving a data discovery task in the Open Data BCN repository, which will be the source from where the data will be collected.

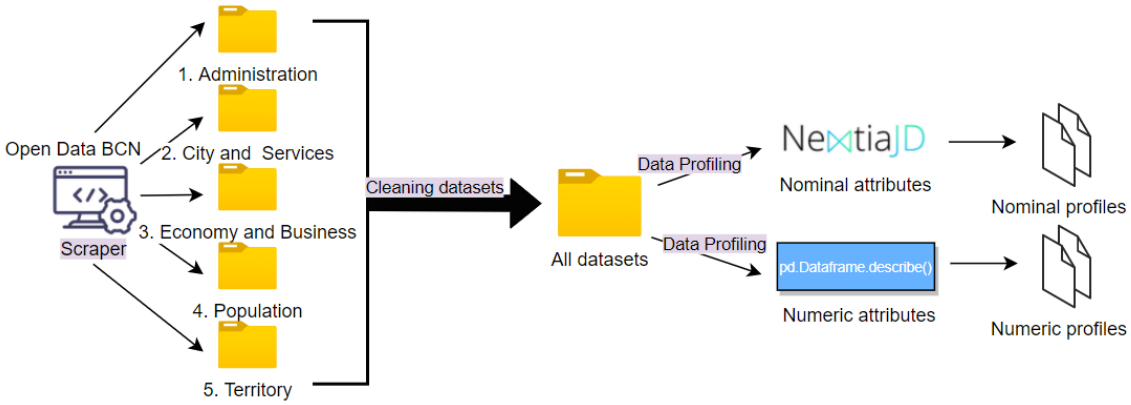


Figure 7: Overview of the Data Collection and Preparation processes

3.1 Data Collection

The very first part of this process is to be able to obtain the datasets from the Open Data BCN repository efficiently. As mentioned in Section 1.4, data is distributed among 5 thematic areas and multiple sub-thematic categories and it is saved under different formats. For our scenario, we will only focus on the five principal thematic areas, denoting them as classes or categories and on tabular data with the `.csv` extension. The reason why we are not considering the sub-thematic areas is because there are not enough representations of each, so it is an impossible task to implement a model that can differentiate them and even less classify the datasets into those sub-areas.

In the main page of the Barcelona's council website, we can access the different thematic areas easily. We show the main page layout in Figure 8.

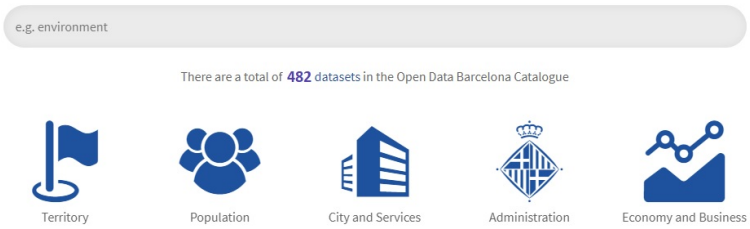


Figure 8: Open Data BCN main page

Once having selected the desired thematic area we access another page of the website that contains links redirecting to the pages in which the datasets are stored under different formats. In Figures 9 and 10 we can see how the Open Data BCN stores the datasets in the aforementioned links.

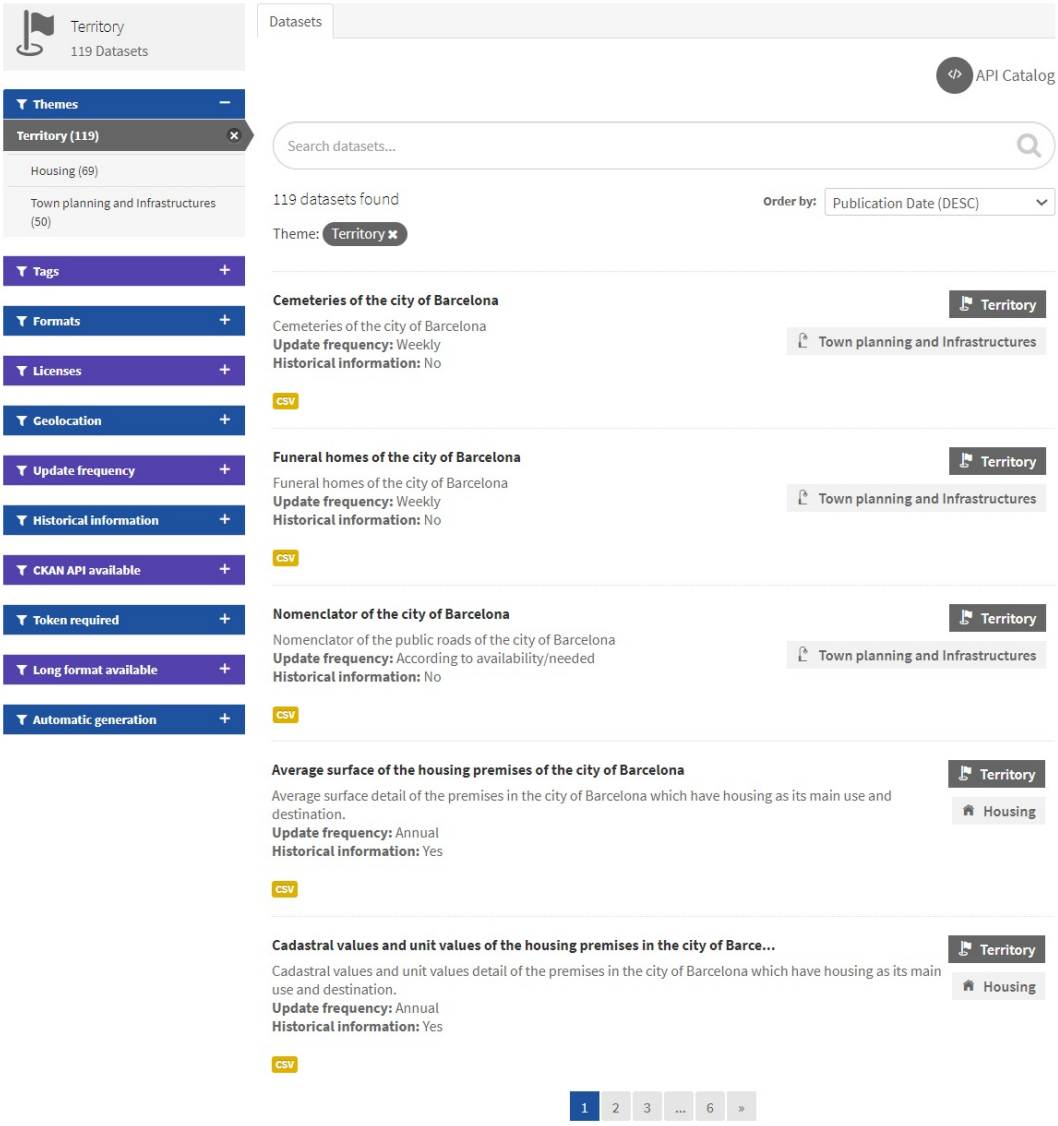


Figure 9: Example of the Open Data BCN distribution for territory thematic area

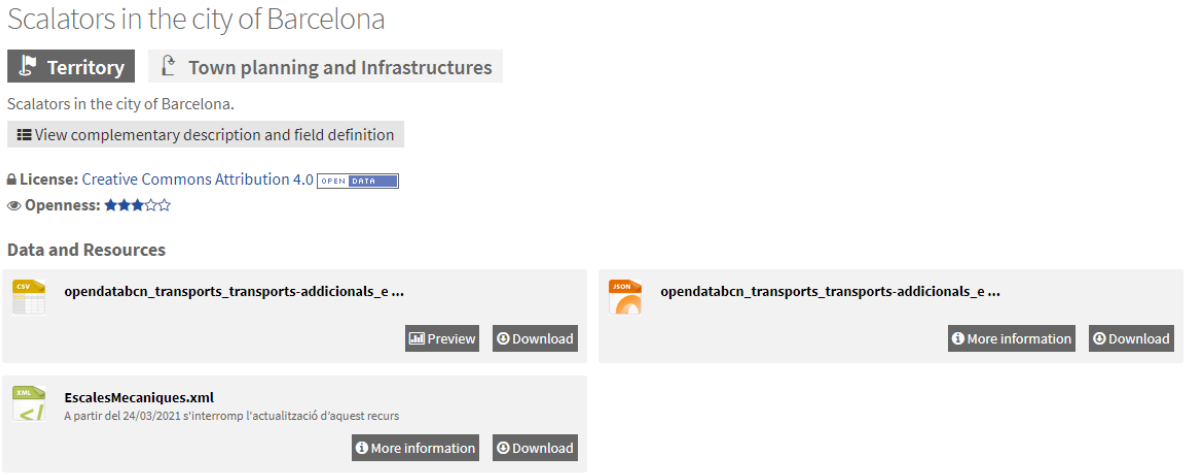


Figure 10: Example of the download site of a specific dataset in Open Data BCN

To obtain all the datasets of this repository manually would mean to access approximately 490 links and download their `.csv` extension files one by one, which would be very time consuming.

3.1.1 Scraper

Most of the time online repositories do not have the option to download their datasets at once, or even less to add the labels of the categories when getting the datasets. This is the case of Open Data BCN, which does not provide an efficient or practical way to extract all its datasets. In order to collect the data, a scraper script, `1.scraper.py` [18] was developed using `python` libraries `bs4, BeautifulSoup`[19] and `urlib.request, urlopen`[20].

The scraper works as follows: For each of the categories it scraps the pages containing download links, identifying the `.csv` file and downloading it into a folder destined to this specific class. The name given to the dataset is the name of the title of the download link, so we can be able to find the dataset in the website if needed. For example, in Figure 10 the `.csv` would be downloaded into a folder named *Territory*, by the name *scalators in the city of barcelona.csv*.

After running the scraper, we obtain the following 5 folders containing the raw datasets fetched directly from the website:

- 1. Administration: 77 datasets.
- 2. City and Services: 97 datasets.
- 3. Economy and Business: 15 datasets.
- 4. Population: 90 datasets.
- 5. Territory: 54 datasets.

[18] `https://github.com/dtim-upc/GCNN-Open-Data-BCN/blob/main/Scripts/1.scraper.py`
[19] `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`
[20] `https://docs.python.org/3/library/urllib.request.html`

The next step is to prepare the data, cleaning those datasets and format them if needed.

3.2 Data Preparation

3.2.1 Data Cleaning

The first step of data cleaning is to add the label of the class in each of the datasets name as we will need to put all datasets in the same folder when profiling the data, but still be able to determine each of their class. The script is `2.rename_datasets.py` [21] and simply adds the number of the class at the beggining of the dataset name. For example, following the example of Figure 10, the renamed dataset would be *4_scalators in the city of barcelona.csv*. Once all the datasets are renamed they are moved into a new folder containing all the datasets.

The second step is to clean the column names of the datasets as there are characters that are not supported by the profiling tool that we use for data profiling. The script `3.clean_datasets.py`[22] removes those characters and also replaces white-spaces from the dataset title with underscores.

3.2.2 Data Profiling: Meta-Feature Extraction

For profile-based approaches of data discovery we need to collect different profiling metadata from the dataset. In this case, we collect meta-features from the attributes of each dataset using a system that supports data discovery over data lakes relying on attribute profiles called `NextiaJD`[23] [43]. This system is build on top of `Apache Spark` and provides succinct representations that capture the underlying characteristics of the schemata and data values of datasets, which can be efficiently extracted in a parallel and distributed fashion. Despite being a profile-based approach, `NextiaJD` presents a similar predictive performance to that of hash-based solutions, yet better adapted for large-scale scenarios, while benefiting from linear scalability.

Nevertheless, one of `NextiaJD` limitations is that it only extracts meta-features from nominal attributes. In order to have a complete profile including both nominal and numeric attributes of the datasets, we extract metadata of the numeric columns by using the python `pandas` function `pd.DataFrame.describe()` that generates descriptive statistics of the attributes.

Meta-feature	Description
count	The number of rows of the dataset
mean	Mean value of the numeric attribute
std	The standard deviation of the numeric attribute
min	The minimum value of the numeric attribute
25%	Percentile P_{25} of the numeric attribute
50%	Percentile P_{50} of the numeric attribute
75%	Percentile P_{75} of the numeric attribute
max	Maximum value of the numeric attribute

Table 2: Meta-features extracted with pd.DataFrame.describe() method for numeric attributes

[21] `https://github.com/dtim-upc/GCNN-Open-Data-BCN/blob/main/Scripts/2.rename_datasets.py`
[22] `https://github.com/dtim-upc/GCNN-Open-Data-BCN/blob/main/Scripts/3.clean_datasets.py`
[23] `https://www.essi.upc.edu/~snadal/nextiajd.html`

The meta-features of the nominal attributes that we can extract with NextiaJD are the depicted in Table 3. On the other hand, the set of numeric meta-features extracted by using python are depicted in Table 2.

Category	Meta-feature	Description	Normalized
Cardinalities	Cardinality	Number of distinct values within an attribute	Yes
	Uniqueness	Measures if the attribute contains unique values	No
	Incompleteness	Measures the number of missing values	No
	Entropy	Measures de variety of an attribute	Yes
Value distribution	Average frequency	The average value of the frequency distribution count	Yes
	Min frequency	The minimum value of the frequency distribution count	Yes
	Max frequency	The maximum value of the frequency distribution count	Yes
	SD frequency	The standard deviation of the frequency distribution count	Yes
	Octiles	The octiles (quantiles) of the frequency distribution in percentages	No
	Min perc frequency	The minimum value of the frequency distribution in percentages	No
	Max perc frequency	The maximum value of the frequency distribution in percentages	No
	SD perc frequency	The standard deviation of the frequency distribution in percentages	No
	Constancy	Frequency of the most frequent value divided by number of rows	No
	Frequent words	The 10 most frequent words	No
	Soundex	The 10 most frequent words in soundex representation	No
Syntactic	Data type	The data type of the attribute (i.e. numeric, alphanumeric, alphabetic, nonAlphanumeric, datetime)	No
	Specific type	The specific type of the attribute (i.e. phone, email, url, ip, username or phrases)	No
	Percentage data type	The percentage for each data type detected in the data values	No
	Percentage specific type	The percentage for each specific type detected in the data values	No
	Longest string	The number of characters in the longest string	Yes
	Shortest string	The number of characters in the shortest value in the attribute	Yes
	Average string	Average length of the string in term of characters	Yes
	Number of words	The number of words in the attribute	Yes
	Average words	The average words in the attribute	Yes
	Min words	The minimum words in the attribute	Yes
	Max words	The maximum words in the attribute	Yes
	SD words	The standard deviation in the attribute	Yes
Pair metadata	Best containment	The containment score assuming all distinct values are covered	No
	Flipped containment	Containment assuming all distinct values are covered divided by max cardinality	No
	Name distance	Measures the difference of two attribute names using Levenshtein distance	No

Table 3: Meta-features composing profiles offered by NextiaJD

For NextiaJD to compute all the profiles of the set of datasets a .csv file needs to be created having as rows each of the datasets and as columns: the filename, the delimiter, if it is multi-line, the extension of the file, the file size, a bool ignoreTrailing and the source (path were the dataset is saved). The script `4.create_information_csv.py`[24] iterates over the datasets and inserts a row for each with the mentioned columns. Once this is done, we can run the `scala` (`5.profiling.scala`[25]) script that computes NextiaJD's profiles. The script also drops the meta-features returning non-numerical or non-categorical values because those will not be useful when using the data for modeling (`FrequentWords`, `FrequentWordsInSoundex`, `firstWord`, `lastWord`).

At the end, for each dataset we obtain a nominal profile of dimensions: n x 45, where n is

[24] https://github.com/dtim-upc/GCNN-Open-Data-BCN/blob/main/Scripts/4.create_information_csv.py
[25] https://github.com/dtim-upc/GCNN-Open-Data-BCN/blob/main/Scripts/5.profiling.scala

the number of nominal attributes of the dataset and 45 corresponds to the 43 meta-features extracted by `NextiaJD` that we selected plus a column for the dataset name and another for each of the attributes names. Likewise, numeric profiles are computed by running the script `6.numeric_profiling.py`[26], obtaining files of size m x 11, where m is the number of numeric attributes of the dataset and 11 corresponds to the 9 meta-features we extract plus two columns for the dataset and the attributes names.

3.2.3 Data Preparation as Input for Models

All the processes described in this subsection can be found at `7.create_input_models.py`[27]. In order to use the profiles as the representation of the datasets to train our models we need to finish cleaning them and aggregate them obtaining a vector for each of the profiles to form the model embeddings. The aggregation step is crucial for two reasons: To have embeddings with the same dimension for all the datasets, and to convert the profiles extracted from attribute-level to dataset-level as we are trying to find similarities between tables and not attributes.

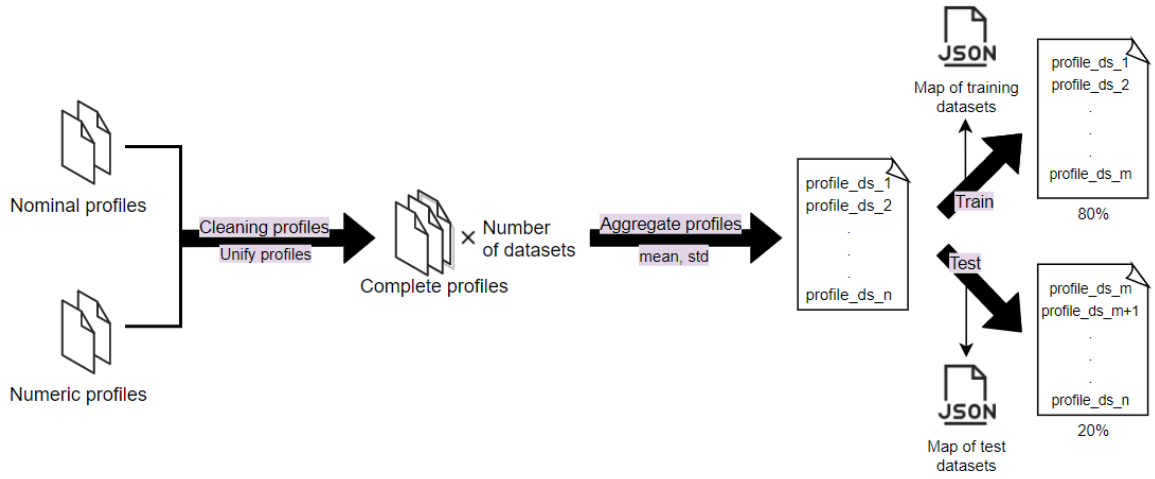


Figure 11: Overview of the Data Preparation as input for the models

The first step is to select the profile attributes that will be used to train the model. As Neural Networks and some ML models can not handle nominal or categorical attributes we apply *one-hot-encoding* transformations to categorical variables (`data_type` and `specific_type`). This step modifies the dimensions of each nominal profile to n x 62 as we replaced 2 categorical variables for 19 *dummy* variables, denoting each of their categories which we obtained when applying the *one-hot-encoding* method.

Then, we unify the nominal and the numeric profiles, forming a single profile for each data set, by performing a left join on those two `csv` files with as key the dataset and the attribute names. We also replace all the missing values by 0 which does not affect the models predictions

[26] `https://github.com/dtim-upc/GCNN-Open-Data-BCN/blob/main/Scripts/6.numeric_profiling.py`
[27] `https://github.com/dtim-upc/GCNN-Open-Data-BCN/blob/main/Scripts/7.create_input_models.py`

negatively because in this context a missing value means that the attribute was empty which can be replaced by having a 0 instead. The left join allows us to create profiles for a dataset even when this one does not have numerical attributes. An inner join would make us loose the profile of the dataset whenever this one would not have either numerical or nominal attributes. On the other hand, a right join would make some of the profiles very sparse, as for those which would not have nominal attributes, 43 columns would be filled with zero values. At the end, we drop the column containing the attribute name as this one will not be relevant for the next steps.

At the same time as we merge the profiles we also aggregate them. The aggregation method consist in computing the mean value and the standard deviation of each of the profile columns for a particular dataset. In other words, we group the profile by their dataset name using as aggregation functions the mean and the standard deviation. Hence, each dataset profile will be a row of twice as many columns as it had before the aggregation step. The aggregation step returns a single `csv` file of dimensions 319 x 140, where 319 is the number of datasets remaining after all the previous processes (those which have nominal attributes) and 140 is the number of meta-features remaining after the aggregation step plus a column containing the dataset name and another column containing the number of the class membership of the dataset which we call the *ground-truth*. In Figure 12 we can see the first rows of the profiles file obtained after all these steps.

	C	A	B	D	E	F	G
1	dataset_name	avgnumberwords_mean	avgnumberwords_std	maxnumberwords_mean	maxnumberwords_std	minnumberwords_mean	minnumberwords_std
2	1_Accions_formatives_adre	3.782407407407408	8.838556500838726	13.222222222222221	31.15596964378487	1.0	2.7838821814150108
3	1_Adreces_per_secció_cen	1.2772362126545718	0.4801872060156881	2.3333333333333335	2.3094010767585034	1.0	0.0
4	1_Base_de_dades_del_pres	3.671798780487805	0.8225328283377131	5.8	1.6865480854231356	1.4	1.07496769977314
5	1_Catàleg_d'estatuts_al_Re	1.7008099640864984	4.175381758307629	13.782608695652174	19.87059585694765	-0.34782608695652173	1.1395396985790713
6	1_Catàleg_d'informes_i_est	1.681491640543365	4.495302677257132	41.020833333333336	94.75331680790543	-0.6875	0.8789803086410546
7	1_Catàleg_d'informes_prese	2.0536012070197724	5.6312663155053695	23.306122448979593	44.68463759207065	-0.42857142857142855	1.0801234497346435
8	1_Catàleg_d'ordenances_fis	2.750226517668378	6.1365583083047195	35.627906976744185	74.68792955829896	0.023255813953488372	1.6400983143153811
9	1_Catàleg_d'ordenances_m	2.7837639151438776	5.256840007978154	39.21739130434783	77.30772939463807	-0.34782608695652173	1.2150425928400448
10	1_Catàleg_del_Repositori_(	0.4021135650182908	2.700280773280076	69.56338028169014	164.43650380221587	-0.9436619718309859	0.3332662575034293
11	1_Catàleg_de_bases_d'ajut	1.9902967321898093	5.532338785683947	18.693877551020407	42.482449357368274	-0.2653061224489796	1.2710807442894418
12	1_Catàleg_de_bases_de__	1.45003465003465	3.681972399024354	11.08108108108108	25.64650547819806	-0.2972972972972973	1.2663347123109292
13	1_Catàleg_de_bases_per_a	3.521877655055226	10.102354452063521	14.818181818181818	33.78886064362158	0.22727272727272727	2.1979232942964586

Figure 12: First columns of the aggregated profiles file

Finally, we need to split our aggregated profiles file, which will be used as input for the models, into training and test sets. For that we keep 80% of the rows to form the training set and we leave 20% for the testing part. It is important to state that the class-balance that we originally had for each of the categories was respected when forming the train and test sets. As models do not support non-numeric attributes we drop the column of the dataset name and we generate a dictionary stored in a `JSON` file, called *Map of datasets* containing the information of which row contains which profile. The dictionary has as key the number of the row (starting with a 0 for the first profile row) and as value the name of the dataset contained by the row. For example in Figure 12 in the dictionary we would have the pairs: (0, 1_Accions formatives adreçades al personal municipal de la ciutat de Barcelona.csv), (1, 1_Adreces per secció censal de la ciutat de Barcelona.csv), and so on.

4 Overview of Classification Approaches Proposed for Barcelona Open Data Discovery

In order to answer the research questions formulated in Section 1.5.2, we need to develop classification approaches using traditional ML and DL techniques that will serve as baselines to compare with the GNN approach to evaluate its performance.

In this section, all the datasets have been normalized by mean and standard deviation using the `sklearn.preprocessing.StandardScaler()` before feeding the models. Also, all the choices of the hyper-parameters of the different models have been made maximizing the accuracy metric, which for classification problems is simply the rate of correct classifications.

4.1 Classification using traditional ML

The similarity data discovery task we are trying to solve is reduced to categorizing the datasets with their class membership, hence, we face a classification problem. In Machine Learning, models like Classification Trees, Support Vector Machine's (SVMs) and Logistic Regression (LR) are arguably the most popular approaches for classification problems. These traditional models learn in a supervised way due to the availability of class labels.

4.1.1 Logistic Regression (LR)

Logistic regression is a statistical model that in its basic form uses a logistic function[28] to model a binary dependent variable, although it can be adapted for non-binary dependent variables in order to solve multi-class classification problems like in our case.

The model used is the `LogisticRegression` provided by `sklearn.linear_model`. Around 20 different configurations of this model were tried out by changing its hyper-parameter values[29]. The best set of hyper-parameters chosen for this model, after a comparison-search using accuracy metric as the goodness of fit, are:

- *penalty* = **l2**. The penalty is used to specify the norm used in the penalization of the model.
- C = **0.15**. This coefficient corresponds to the inverse of the regularization strength. Smaller values specify stronger regularization.
- *solver*: **newton-cg**. This hyper-parameter designes the algorithm to use in the optimization problem.
- *max_iter* = **1000**. Which is the maximum number of iterations taken for the solver to converge.
- *multi_class* = **multinomial**. Specifies that the loss minimised is the multinomial loss fit across the entire probability distribution.

[28] It is a common S-shaped curve (sigmoid curve). See: `https://en.wikipedia.org/wiki/Logistic_function`

[29] All possible hyper-parameters can be found here: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

- All other hyper-parameters are set to the default value.

4.1.2 Random Forest (RF)

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. When dealing with a classification problem, it operates by constructing a multitude of decision trees at training time and outputting the class that was assigned in more instances.

The model used is the `RandomForestClassifier` from the `sklearn.ensemble`. Around 25 different configurations were tried out for this model by changing the hyper-parameter values[30] of the model. The best hyper-parameter set found is the following:

- *max_depth*: **10**. Sets the maximum depth of the trees.
- *n_estimators*: **100**. Which determines the number of trees in the forest.
- *max_features*: **sqrt**. Sets the rule to apply to the number of features to consider when looking for the best flip. If sqrt, then $max\_features = sqrt(n\_features)$.
- *random_state*: **0**. This parameter controls the randomness of the bootstrapping of the samples used when building the trees.
- All other hyper-parameters are set to default value.

4.1.3 Suport Vector Machines (SVM's)

Support vector machines are a set of supervised learning methods used for classification, regression and outliers detection. As we are dealing with a classification problem, the SVM to use is a Support Vector Classifier (SVC). The objective of an SVC is to fit to the data provided, returning a "best fit" hyperplane that divides, or categorizes it. After getting the hyperplane, some features can be fed to the classifier to see what the predicted class is.

The model used is `SVC` from `sklearn.svm` python library. Around 15 different configurations of hyper-parameters[31] were tried out for this model and the best hyper-parameters fitting the model to our data are:

- *C*: **150**. Designing the regularization parameter, i.e. the strength of the regularization is inversely proportional to this parameter C.
- *decision_function_shape*: **ovo**. Multi-class strategy for the decision function of classification.
- *shrinking*: **False**. Sets whether to use the shrinking heuristic or not.

[30] All possible hyper-parameters can be found here: `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

[31] All possible hyper-parameters can be found here: `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`

- *probability*: **True**. Whether to enable probability estimates and it internally uses 5-fold cross-validation.
- All other hyper-parameters are set to default value.

4.2 Classification using Neural Networks (NN's)

Another possibility to approach a multi-class classification problem is using Deep Learning. In this case, we will model a simple neural network called Multi-Layer Perceptron (MLP). MLP is a supervised learning algorithm that given a set of features and a target, can learn a non-linear function approximator for either classification or regression. There are at least three node levels in it: an input layer, a hidden layer, and an output layer. Each node, with the exception of the input nodes, is a neuron with a nonlinear activation function. For training, it employs backpropagation, a supervised learning approach.

4.2.1 Multi-Layer Perceptron (MLP)

For this method we use the `MLPClassifier` offered by the `sklearn.neural_network` python library. After around 25 different model configurations changing the hyper-parameter values[32], we chose the following set of hyper-parameters, which are also depicted in Figure 13, as the best possible configuration based on the accuracy metric:

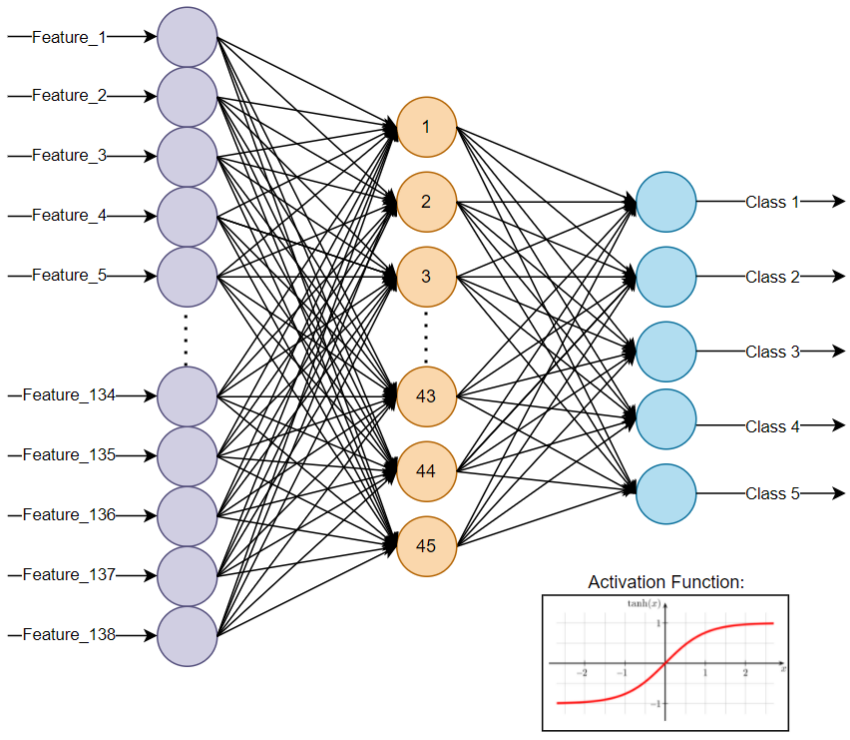


Figure 13: Schema of MLP-1 model architecture

[32] All possible hyper-parameters can be found here: `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`

- *hidden_layer_sizes*: **45**. Representing the number of neurons in the hidden layer.
- *activation*: **tanh**. Designing the activation function used for the hidden layer. In this case, it is the hyperbolic tangent function.
- *alpha*: **0.01**. The L2 penalty parameter (regularization term).
- *random_state*: **2**. Integer set for reproducible results across multiple function calls.
- All other hyper-parameters are set to default value.

5 Overview of the Link Prediction - GCNN Approach Proposed for Barcelona Open Data Discovery

This approach is modeled using the python library `Pytorch Geometric`. As in Section 4, data was normalized using the `sklearn.preprocessing.StandardScaler()` before feeding the Graph Neural Network.

5.1 Choice of Link Prediction

As seen in Section 2.2.4, there are many different tasks we can solve with a GNN. In the context of this thesis and following the definition of the problem of finding similar datasets among the Open Data BCN repository (Section 1.4.2), we need to find the proper way to map the similarity task to a task that can be solved within a graph. Hence, we need to define nodes, node embeddings and edges in a way that it is possible to model a GCNN to predict similarities among datasets by finding out which datasets share the same class. We decide to represent the datasets with nodes and the profiles extracted from them as node embeddings. As for what concerns the edges, we need to find a criterion to link the datasets in a meaningful way. For that, we define an edge between two nodes as an indicator of those two nodes being similar, and hence, belonging to the same class. Therefore, finding similarities among new datasets of the repository would mean to find new edges between the nodes representing the new datasets and the nodes we already have in the graph. This is defining a link prediction task. Choosing this approach, we leverage the message passing framework which takes into account the information of the neighbouring nodes to influence each of the graph nodes. In this way, we enable the GCNN to learn node features that encode the different properties of each class. The latter is achieved by using both the node attributes and edge structure information to construct the node feature representation. Contrarily, if we wanted to use a node classification framework for this problem, we would not have a consistent criterion to link the nodes and therefore we would not be able to learn node embeddings that shared their properties among the other nodes that belong to the same class.

5.2 Input Graph Representation

As mentioned above we represent the Open Data BCN repository as a graph by codifying the datasets as nodes, the profiles as node embeddings and the class memberships as edges. The idea is to transform a multi-class classification problem as we solved in Section 4 into a link prediction task using a GCNN. With this goal we model the problem as a GCNN receiving as input the graph from which we want to find similarities and predicting the similarities between the nodes by finding edges and their corresponding probability (we decide later on which threshold to use on the probability value returned by the model in order to predict whether it is high enough or not to consider the edge valid). As each edge determines the fact of two nodes sharing the same class, we can model a GCNN to understand the relations between datasets leveraging the message passing principle.

For that, we need a graph to train the model and another graph, which will be an extension of

the training graph, to predict links for each of the new nodes that will enter the repository.

5.2.1 Input Graph for Training

In this modeling the input graph that the GCNN receives for training is formed by as many fully-connected components as classes we have in the conventional classification problem. In the schema depicted in Figure 14 we see a small example of the input graph modeled for our problem, where each class is represented by a different color.

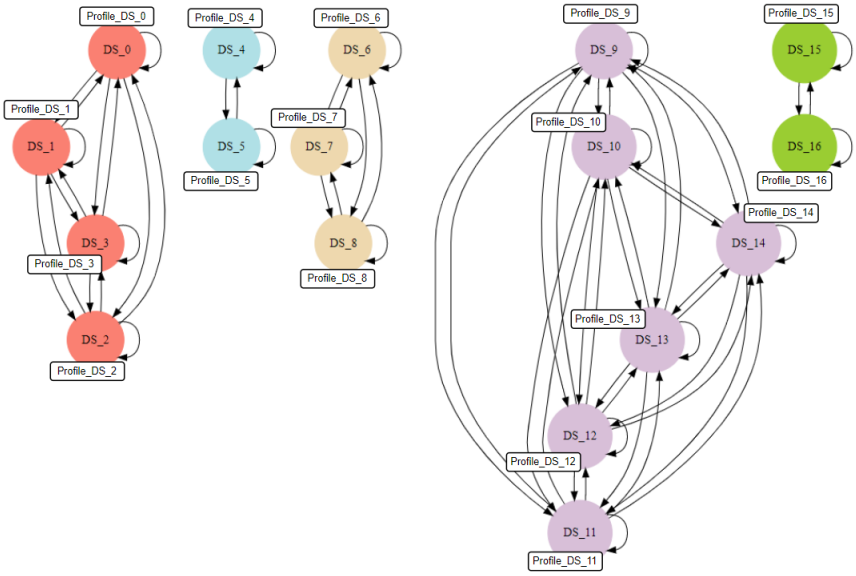


Figure 14: Example of input graph for training step GCNN - Link prediction

As input, the model needs to receive the information to create the graph. What we provide to the model is:

- a list with the profiles ordered by the node tag (being this one the integer assigned during the data preparation process in the mapping step).
- a list of all existing edges in the input graph (if a pair $(node\_0, node\_1)$ appears in the list of edges, it means that there exist an edge between $node\_0$ and $node\_1$).

The input graph of this training step is made taking only the datasets of the Open Data BCN repositories belonging to the train set that we created in Section 3.2.3. To provide only one graph to the model for the training step is enough because link prediction approaches use a train-test split of the edges of one graph in order to be trained. They train themselves by masking sets of edges and trying to predict them in multiple iterations.

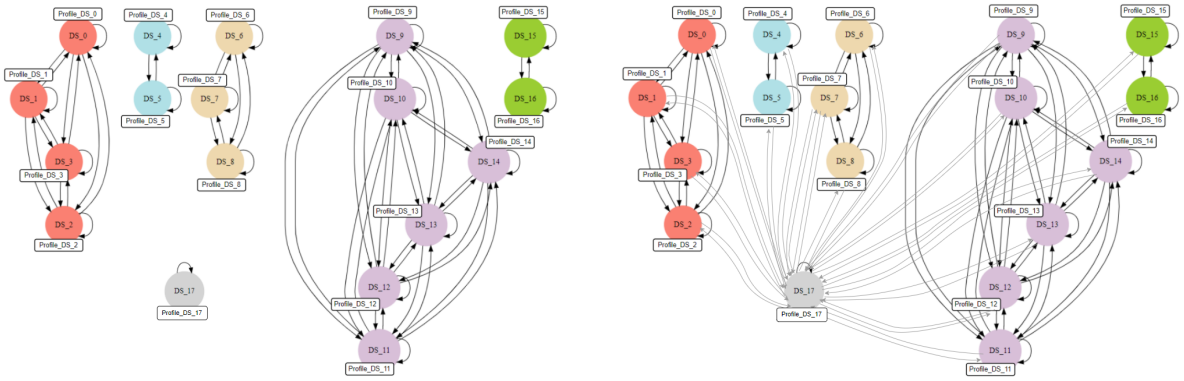
5.2.2 Input Graph for Testing

Once the model is trained for the graph representing the entire training set of the Open Data BCN repository, we need to create another graph including the testing datasets for the model to predict new edges for them.

The idea is to pass to the model a new graph including all the datasets of which we already know the class (the ones used in the train step) and an additional test dataset for which we want to predict the class. The test step will be performed in as many iterations as datasets we have in the test set. The known datasets will be exactly codified as in Section 5.2.1 and for the test datasets we try two different approaches.

Approach 1: Model adds edges. The test dataset is codified as a disconnected node with its profile as embedding (see Figure 15 (a)). The model predicts the edges between this node and the rest of the graph, finding similarities between them. Then we define a metric that computes the class of the test node taking into account the class of the nodes it linked to, and their corresponding probabilities.

Approach 2: Model removes edges. The test dataset is codified as a node, with its profile as embedding, which is connected to all the other nodes of the graph (see Figure 15 (b)). The model needs to "disconnect" the test node of the nodes that aren't similar to the dataset (we understand as "disconnect" to not predict the edges having a probability lower than a given threshold). Then we define a metric that computes the class of the test node taking into account the class of the nodes it is still linked to, and the corresponding probabilities.



(**a**) Input Graph Test, Approach 1

(**b**) Input Graph Test, Approach 2

Figure 15: Examples of input graph for test step of GCNN Link prediction. The test node is represented as a grey node because its class is unknown. On the right, the test node is unlinked from the other components of the graph. On the left, the test node is linked to all the other nodes of the graph.

5.3 Model Architecture

The structure of the GCNN architecture for this approach is depicted in Figure 16. It is formed by three graph convolutional layers with ReLU activation functions and dropout layers in between. After the output of the last convolutional layer, a softmax layer is added in order to normalize the output of the network to a probability distribution. The layout of the model presented above is achieved after trying out a large amount of different configurations and keeping the one obtaining best results (in terms of loss and accuracy).

A decoder transforms the model output into a score which is a probability, thanks to the softmax layer, associated to each pair of nodes in the graph. Then, another decoder (called `decoder_all`)

transforms the probabilities into an adjacency matrix, based on a probability threshold (if the probability of the pair of nodes returned by the model is greater or equal than the threshold, then in the adjacency matrix the edge will be present, otherwise, we won't consider it). We also define a function `obtain_probs` that returns for a pair of nodes the probability predicted by the model so we can use it later on.

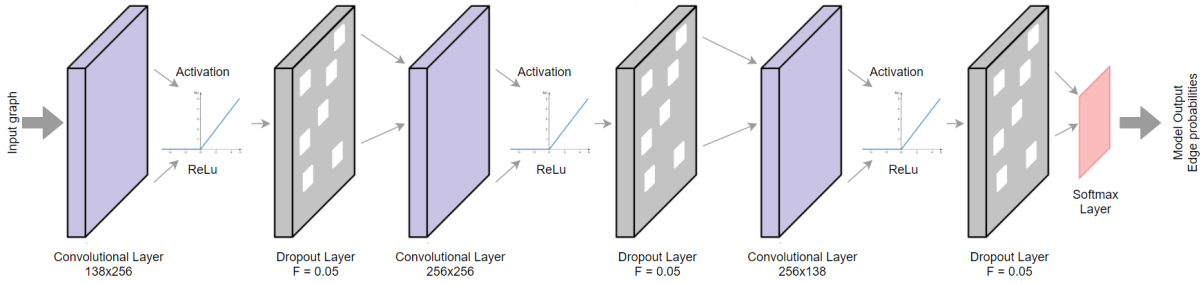


Figure 16: GCNN model architecture

More specifically, the hyper-parameters of the GCNN giving the best results are the following:

- *Number and sizes of layers*: Three layers of sizes (138x256), (256x256), (256x138).
- *Drop out factor*: **0.05**. During training, probability of randomly set to zero some of the elements of the input tensor.
- *Activation function*: **ReLU**. Activation function used between layers. Decides, whether a neuron should be activated or not by using the ReLU function (piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero: $y = max(0, x)$).
- *Optimizer*: **Adam**. Optimization algorithm used to change the attributes of the neural network such as weights and learning rate to reduce the losses.
- *Learning rate*: **0.01**. Controls how much to change the model in response to the estimated error each time the model weights are updated.
- *Epochs*: **40**. Number of times that the learning algorithm will work through the entire training dataset.

Moreover, as mentioned before, the model outputs probabilities and not a binary decision for whether two nodes are linked or not, hence, we define a probability threshold in the `decoder_all` as a supplementary hyper-parameter. This threshold is set to 0.5, meaning that if the probability of connection between two nodes is greater than 0.5, the model predicts an edge between those two nodes.

The loss function minimized by the model is the `binary_cross_entropy` implemented in the pytorch's `functional` package. It is a function that measures the *binary cross entropy* between the target and the output probabilities. The binary cross entropy evaluates each of the predicted

probabilities against the actual class output, which can be 0 or 1. This function penalizes the probabilities based on their deviation from the expected value. It follows the bellow equation:

$$Binary\_Cross\_Entropy = -(ylog(p) + (1-y)log(1-p))$$

where:

- y is the binary indicator (0 or 1) of whether a pair of nodes has an edge or not
- p is the predicted probability for a pair of nodes to have an edge or not.

5.4 Link prediction to Classification

As mentioned in the previous section, we obtain as output of the model after the decoding steps: an edge-adjacency matrix and the corresponding probabilities of each of the edges defined in the matrix. Nevertheless, what we are trying to solve is a classification problem, so we need to adapt this output so that we can obtain the class assigned to each node.

To do so, we need to take into account both,

- the number of edges to a node having a specific class and
- the probabilities of each of those edges.

Hence, we will define a function that, given the edges of a node and its probabilities, returns the class of this node.

5.4.1 Class Selector Function

Before defining the class selector function we need to define other functions that will be needed for its construction.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with $\mathcal{V} : v_1, \ldots, v_r$ nodes and $\mathcal{E} : e_1, \ldots, e_s$ edges. Let $\mathcal{C} : \{c_1, \ldots, c_m\}$ be the set of possible classes that a node of the graph can belong to.

We take an arbitrary node v_i. Let be $\mathcal{E}_i : e_{i_1}, e_{i_2}, \ldots, e_{i_n}$ the set of edges of the node v_i linking with nodes that belong to the class c_l, with their corresponding probabilities $\mathcal{P}_i : p_{i_1}, p_{i_2}, \ldots, p_{i_n}$ and t the threshold probability defined in order for the model to consider an edge between two nodes.

- Definition 1. We define a function that returns a discretization set of the probability set $\mathcal{P}_i$ with as many levels as desired (n_levels sets the number of levels we want), as:

$$discretize(\mathcal{P}_i, n\_levels) \rightarrow \{1, 1 - \frac{1-t}{n\_levels}, 1 - 2 \cdot \frac{(1-t)}{n\_levels}, \ldots, t\}$$

which can also be written as n_levels intervals as follows:

$I_0 = \left[1, 1 - \frac{1-t}{n\_levels}\right]$,

$I_1 = \left(1 - \frac{1-t}{n\_levels}, 1 - 2 \cdot \frac{(1-t)}{n\_levels}\right]$,

...

$I_{n\_levels-1} = \left(1 - (n\_levels - 1) \cdot \frac{(1-t)}{n\_levels}, t\right]$

Note: It is important that the intervals are defined in decreasing order forming a totally-ordered set: $I_0 > I_1 > \cdots > I_{n\_levels-1}$.

- Example 1. In our practical case, we determined $t = 0.5$ and $n\_levels = 5$ obtaining:

$discretize(\mathcal{P}_i, 5) = \{1, 0.9, 0.8, 0.7, 0.6, 0.5\}$, because $\frac{1-t}{n\_levels} = \frac{1-0.5}{5} = 0.1$. And hence, our 5 intervals are: $I_0 = [1, 0.9], I_1 = (0.9, 0.8], I_2 = (0.8, 0.7], I_3 = (0.7, 0.6], I_4 = (0.6, 0.5]$.

- Definition 2. Let $p \in [1, t]$ be a probability and D be a discretization set $D = \{I_0, I_1, \ldots, I_{n\_levels-1}\}$ we define $idx\_interval(p, D)$ as the function returning the index of the interval of D where p belongs:

$$idx\_interval(p, D) \to k, \quad if \;\; p \in I_k$$

- Definition 3. For a node v_i of a given graph $\mathcal{G}$ and a specific discretization set D that has been associeted to categorize the probabilities of the edges of this graph, we define the weight of the class c_l as:

$$Weight\_class(c_l, v_i) = \sum_{j=1}^{n} \left(1 - \frac{idx\_interval(p_{i_j}, D)}{n\_levels}\right) \cdot p_{i_j}, \quad \forall e_{i_j} \in c_l$$

where: $e_{i_j} \in c_l$ stands for the edge e_{i_j} connecting the node v_i to a node belonging to the class c_l.

- Definition 4. For a given graph $\mathcal{G}$, we define the $counter\_class(\mathcal{G}, c_l)$ function as the function that returns the amount of nodes of the graph belonging to a given class c_l:

$$counter\_class(\mathcal{G}, c_l) \to |c_l|_{\mathcal{G}}.$$

where $|c_l|_{\mathcal{G}}$ is the cardinal of the class c_l in $\mathcal{G}$.

- Definition 5 ($Class\_Selector$). For a given node v_i of a graph $\mathcal{G}$ for which we want to predict the class, we define the $Class\_Selector$ function as the function that returns the class that maximizes the normalization of the $Weight\_class$ function for this given node.

$$Class\_Selector(v_i) = \arg\max_{c_l \in \mathcal{C}} \left(\frac{Weight\_class(c_l, v_i)}{counter\_class((\mathcal{G}, c_l)}\right)$$

We normalize with the $counter\_class(\mathcal{G}, .)$ because the maximum value that the $Weight\_class(., .)$ can return is the same as nodes of the class we have in the graph. This happens when the node is connected to all the nodes belonging to the class with probability equal to 1.

Let's see an example of the $Class\_Selector$ function to clarify the concepts exposed above.

- Example 2. We have a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ which is an outcome of the GCNN link-prediction model. The graph has $|\mathcal{V}| = 14$ with undirected edges and 3 different classes each node can belong to: c_{orange}, c_{purple} or c_{blue}. The number of nodes belonging to each class is: $|c_{orange}| = 5$, $|c_{purple}| = 4$ and $|c_{blue}| = 4$. The classification set defined for this example (and for the whole thesis) is the one depicted in the *Example 1*: $D = \{1, 0.9, 0.8, 0.7, 0.6, 0.5\} = \{I_0 = [1, 0.9], I_1 = (0.9, 0.8], I_2 = (0.8, 0.7], I_3 = (0.7, 0.6], I_4 = (0.6, 0.5]\}$, and hence the probability threshold $t = 0.5$ and $n\_levels = 5$. There is a node that needs to be classified in one of the 3 possible classes and that has been linked by the model to some other nodes with the corresponding probabilities. The example is depicted in Figure 17. For simplicity, only the edges connecting the node we want to classify are shown.

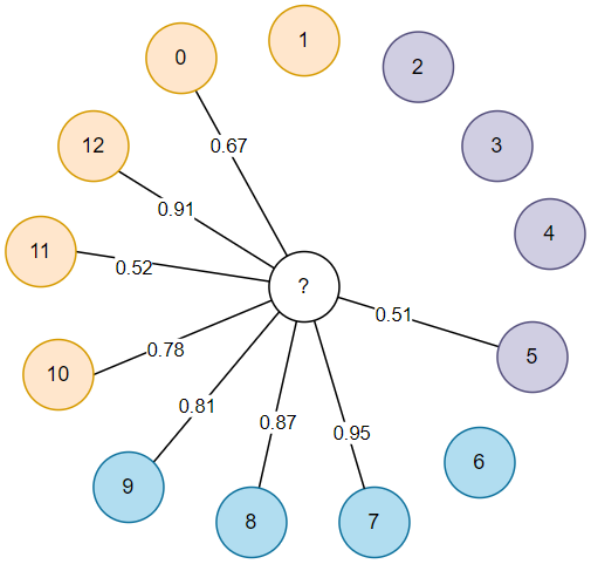


Figure 17: Graph representing the *Example 2*. It simulates the outcome of a GCNN for the testing phase in which we want to predict the class of the white node. Each edge has a corresponding probability, if there is no edge it means that its associated probability was lower than the threshold t and hence, the edge could not be considered as valid.

To determine the class of the node in the center (v_{13}) we apply the $Class\_Selector$ function to that node. We start computing the $Weight\_class(.,.)$ for node v_{13} and for each of the possible classes:

- For class c_{purple} we obtain:

$$\begin{aligned} Weight\_class(c_{purple}, v_{13}) &= \left(1 - \frac{idx\_interval(0.51, D)}{n\_levels}\right) \cdot 0.51 \\ &= (1 - \frac{4}{5}) \cdot 0.51 \\ &= 0.102 \end{aligned}$$

- In the same way, for class c_{orange} we have:

$$\begin{aligned} Weight\_class(c_{orange}, v_{13}) = & \left(1 - \frac{idx\_interval(0.67, D)}{n\_levels}\right) \cdot 0.67 + \\ & + \left(1 - \frac{idx\_interval(0.91, D)}{n\_levels}\right) \cdot 0.91 + \\ & + \left(1 - \frac{idx\_interval(0.52, D)}{n\_levels}\right) \cdot 0.52 + \\ & + \left(1 - \frac{idx\_interval(0.78, D)}{n\_levels}\right) \cdot 0.78 \\ & = (1 - \frac{3}{5}) \cdot 0.67 + (1 - \frac{0}{5}) \cdot 0.91 + (1 - \frac{4}{5}) \cdot 0.52 + \\ & + (1 - \frac{2}{5}) \cdot 0.78 \\ & = 1.75 \end{aligned}$$

- Analogously, for class c_{blue} we obtain:

$$\begin{aligned} Weight\_class(c_{blue}, v_{13}) &= (1 - \frac{0}{5}) \cdot 0.95 + (1 - \frac{1}{5}) \cdot 0.87 + (1 - \frac{1}{5}) \cdot 0.81 \\ &= 2.294 \end{aligned}$$

Next we need to normalize those weights with the cardinals of each class. In this case:

$$\frac{Weight\_class(c_{orange}, v_{13})}{|c_{orange}|} = \frac{1.75}{5} = 0.35$$

$$\frac{Weight\_class(c_{purple}, v_{13})}{|c_{purple}|} = \frac{0.102}{4} = 0.0255$$

$$\frac{Weight\_class(c_{blue}, v_{13})}{|c_{blue}|} = \frac{2.294}{4} = 0.5735$$

Hence, the $Class\_Selector(v_{13}) = c_{blue}$.

6 Results and Discussion

6.1 Baseline methods Results and Evaluation

6.1.1 Train and Test Accuracy

To evaluate the models developed in Section 4 we show a table with the accuracy metrics obtained for the best models of each kind. The train and test sets are the ones that were determined in Section 3.2.3, and hence, are the same for each of the models that we want to evaluate.

Model	Hyper-parameters	Train Accuracy	Test Accuracy
LR	C = 0.15, max_iter = 1000, penalty = l2, solver = newton-cg, multi-class = multinomial	82.35%	73.44%
RF	**max_depth = 10, n_estimators = 100, random_state = 0, max_features = sqrt**	**100%**	**87.50%**
SVM	decision_function_shape = ovo, C = 150, kernel = linear, shrinking = False, probability = True	98.82%	71.88%
MLP	max_iter = 1000, hidden_layer_sizes = 45, random_state = 2, activation = tanh, alpha = 0.01	98.43%	76.56%

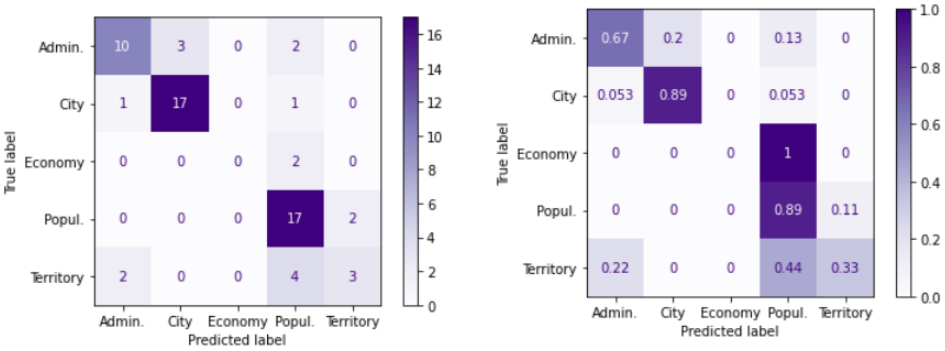
Table 4: Results of accuracy metric for both train and test set of the models defined in section 4.

As we can see in Table 4, the model offering higher performances in terms of accuracy is without any doubt the Random Forest, achieving a score of 100% of accuracy for the train set and 87.5% for the test set. Although the results are pretty good, we can state a slight over-fit in the model, as the train accuracy is significantly above the test accuracy. In fact, we can see this happening for all of the models. An over-fitted model is a statistical model that contains more parameters than can be justified by the data. Because the criterion for picking the model differs from the criterion for judging the model's appropriateness, there is a risk of over-fitting. For example, a model may be chosen based on how well it performs on a set of training data, but its appropriateness may be evaluated by how well it performs on unknown data. When a model is over-fitted we say that it is beginning to "memorize" training data instead of "learning" to generalize from a trend. In our case, what causes the over-fitting is the fact of not having enough data to train the models, and hence, they may be starting to memorize the few instances of data they got and failing generalizing. Nevertheless, being a 5-class classification problem, the test results obtained for all of the models are still high and we can say that the models are learning.

6.1.2 Confusion Matrix

In order to see the classification performances for each of the classes we computed the confusion matrices of all the models so we could analyse them. A confusion matrix is a special table arrangement that permits visualization of the performance of an algorithm. The examples in an actual class are represented by the rows of the matrix, whereas the instances in a predicted class are represented by the columns.

- Logistic Regression:



(**a**) LR Confusion Matrix (**b**) LR Normalized Confusion Matrix

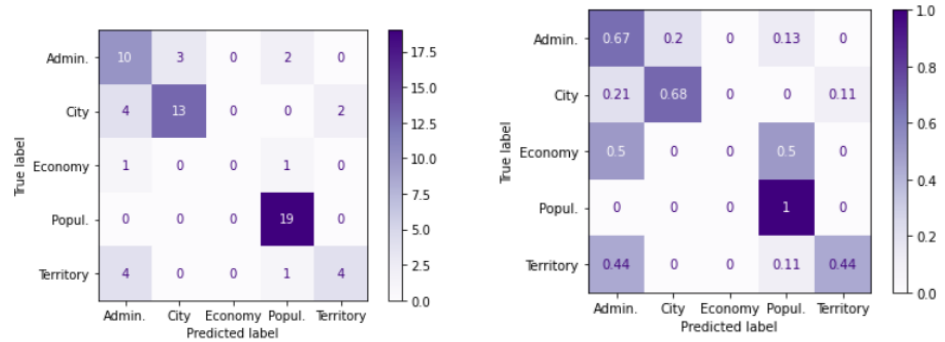
Figure 18: Confusion matrices computed with the predictions made by the Logistic Regression model for the Test set.

- Random Forest:

(**a**) RF Confusion Matrix (**b**) RF Normalized Confusion Matrix

Figure 19: Confusion matrices computed with the predictions made by the Random Forest model for the Test set.

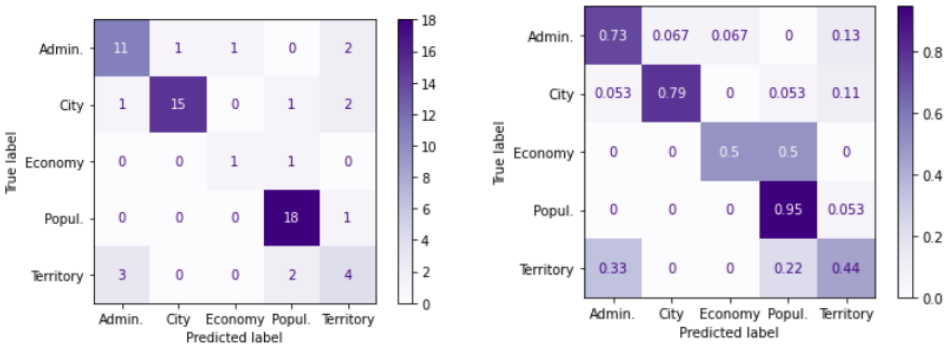
- Support Vector Machines:



(**a**) SVMs Confusion Matrix (**b**) SVMs Normalized Confusion Matrix

Figure 20: Confusion matrices computed with the predictions made by the SVMs model for the Test set.

- Multi-Layer Perceptron:



(**a**) MLP Confusion Matrix (**b**) MLP Normalized Confusion Matrix

Figure 21: Confusion matrices computed with the predictions made by the Multi-Layer Perceptron model for the Test set.

As we can see, all the models have difficulties predicting correctly the samples belonging to the classes: *Economy and Business* and *Territory*, which as we stated in section 3.1.1 are the ones having less datasets. On the other hand, the classes obtaining the highest accuracy performance are *City and Services* and *Population*, obtaining 100% of accuracy in multiple occasions. Those classes are the ones which are the most represented in the Open Data BCN repositories and it is a good thing that the models know how to classify them correctly. In the future we should add in the train set more representations of the other classes if we want a model that is no biased towards any category.

6.2 GCNN approach Results and Evaluation

6.2.1 Results Link Prediction

As explained in Section 5 before tackling the classification problem we model a GCNN that is capable of predicting edges between nodes of a graph. Hence, the first step to validate the approach is to evaluate the link prediction model for the graph representing the Open Data BCN repository (the train datasets only).

Epoch	Loss	Validation Accuracy	Test Accuracy
1	0.6931	0.5	0.5
3	0.6681	0.8900	0.8959
5	0.555	0.9450	0.9451
15	0.5228	0.9936	0.9968
31	0.5181	1.0	1.0

Table 5: Results of the GCNN model as a Link Prediction approach (before transforming it into a Classification approach). We show only the epochs providing results significantly different from its previous epoch.

As the Table 5 shows, the model has learnt to predict all the edges in the graph with a perfect score of accuracy. In Figure 22 we can see both input and output graphs of the model.

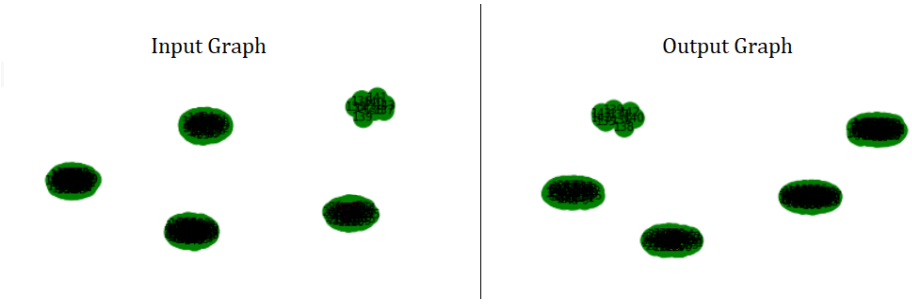


Figure 22: Comparison between the training input graph passed to the GCNN model against the output obtained after the training step. The graphs are displayed randomly, which is why the two graphs look different, but in reality they are the same. The exact same edges are predicted obtaining 100% of accuracy. Each connected component represents a class.

6.2.2 Results Classification

The next step is to transform the adjacency-matrix obtained as output of the GCNN model into predictions of classes for each node. We do that using the $Class\_Selector(.)$ function defined in Section 5.4.1.

	Train Accuracy	Test Accuracy
Approach 1	**100%**	**60.94%**
Approach 2	100%	6.25%

Table 6: Results of the GCNN model as a Classification problem. These are obtained after applying the $Class\_Selector(.)$ function. In the approach 1, the model adds edges to the test nodes and in approach 2 the model removes edges of the test nodes.

The results obtained by the Approach 2 are extremely bad, proving that this approach is clearly not the way to go. Approach 1, although being over-fitted, offers a better performance. Nevertheless, the test accuracy obtained is significantly lower than both the train accuracy and the test accuracy obtained by the methods in 6.1.1.

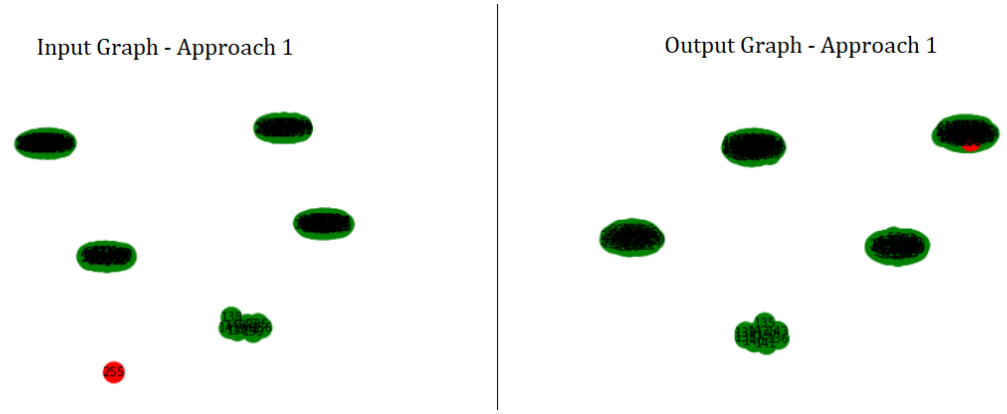


Figure 23: Comparison between the input graph of the GCNN model (approach 1) for a specific test dataset against the output graph obtained from the model. The test dataset is represented as the red node. The node was linked to all the nodes (76 nodes) of a class (class 2) which turned out to be its true class (The normalized $Weight\_Class$ computed was 0.987).

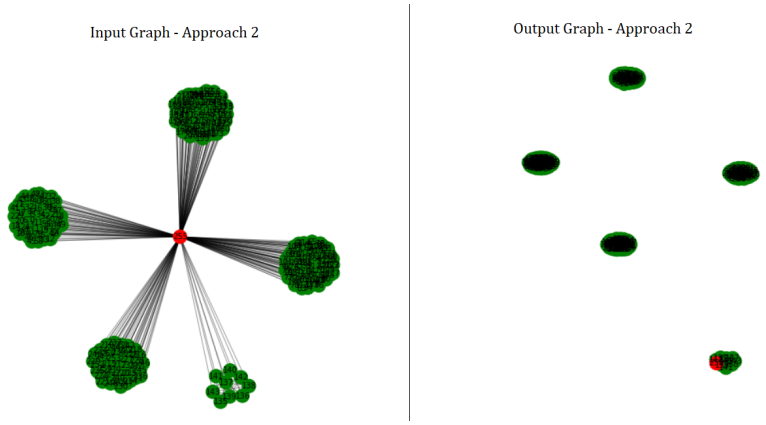
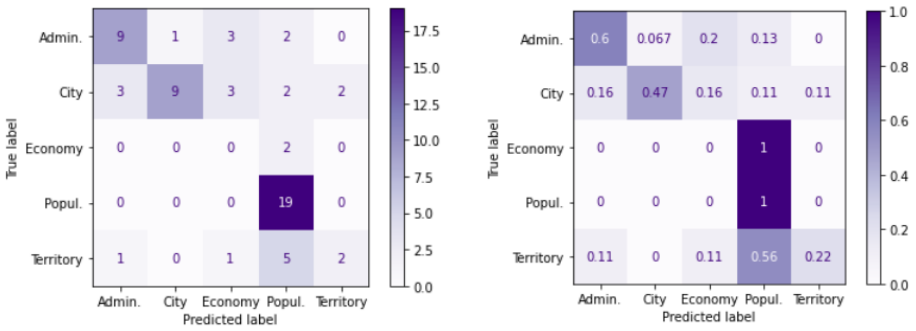


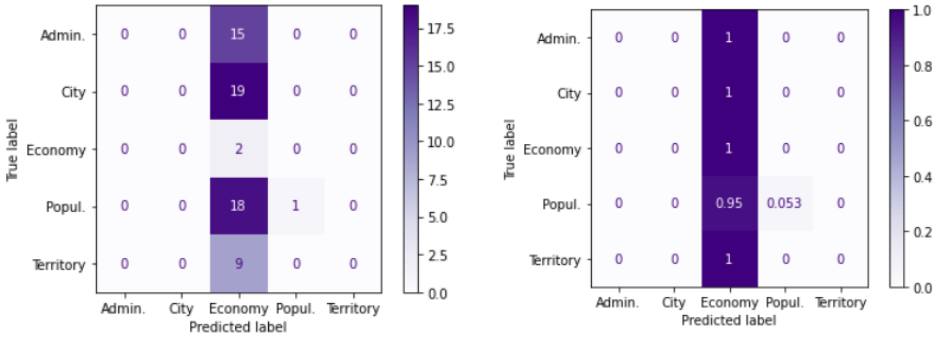
Figure 24: Comparison between the input graph of the GCNN model (approach 2) for a specific test dataset against the output graph obtained from the model. The test dataset is represented as the red node. The model linked it to all the nodes of a class (class 3), but this class is not the true class of the node (the true class is 2). The GCNN miss-classified this test node and the The normalized $Weight\_Class$ obtained was 0.9656.

In order to analyse with more precision the output obtained by the model in both approaches we compute the confusion matrices.



(**a**) Approach 1 Confusion Matrix (**b**) Approach 1 Normalized Confusion Matrix

Figure 25: Confusion matrices computed with the predictions made by the GCNN model for the Test set with the approach 1 as input graph.



(**a**) Approach 2 Confusion Matrix (**b**) Approach 2 Normalized Confusion Matrix

Figure 26: Confusion matrices computed with the predictions made by the GCNN model for the Test set with the approach 2 as input graph.

In Figure 25 we can see that for approach 1 there is a class that the model has learnt to predict with perfect score: the *Population* class. It also classifies *Administration and Services* class with a high score of accuracy. Conversely, the model has difficulties classifying samples belonging to *Territory* and *Economy and Business* categories. These results were expected as those two classes are the ones having the less representation in the train set and in the Open Data BCN repository.

On the other hand, in Figure 26 we notice that the model links all the test nodes but one, to the *Economy and Business* category. Thus makes us think that linking the test node to all the other nodes so that the model could only keep the edges going to/from nodes with its true class is not a good method.

6.3 Discussion

The results of the models presented both in Sections 6.1 and 6.2 are good but they could have been better if we had not encountered some limiting problems.

One of the problems faced involves imbalance classes. In Sections 6.1 and 6.2, we notice that the classes the model encounters more difficulties to classify are the ones with less representation in the Barcelona's repository, showing that the unbalancing of the classes is shrinking the model's accuracy performances. This is an inconvenient that is hard to avoid in real life, because in most data repositories there are no rules about the amount of datasets in each category, and most of the time those will be different. In our case, we knew from the beginning that making in sort that the model could classify correctly the *Economy and Business* thematic area having only 15 available datasets would be an almost impossible task. We could have considered to apply oversampling methods to our profile dataset, but when constructing and trying out the ML models in section 4, we tried a very similar approach using the *class_weight*[33] hyper-parameter provided by those models, and the results obtained were actually worse because the accuracy decreased.

Another limitation comes with the subjectivity of the thematic areas. The dataset categorization in the Open Data BCN repository is a manual task made by different people which decide, sort of subjectively, where to classify the datasets. Hence, there are no specific rules determining whether a dataset should belong to a class or another, and some datasets could have been classified elsewhere if someone different had had to do the task. This adds complexity to the classification problem as it introduces a certain randomness in the dataset categorization.

Furthermore, an additional constraint is added when aggregating the profiles. Although this step is compulsory because we need to obtain profiles at a dataset level and not at an attribute level, it makes us loose a lot of information that affects negatively on the models performances. We had to decide which aggregation technique to use, and we chose to aggregate the data in a Gaussian form by keeping the mean and standard deviation values of the profile meta-features.

[33]We can tweak the existing training procedure to accommodate for the classes skewed distribution. This is achieved by giving different weights to both the majority and minority classes using the class_weight hyper-parameter during the training phase.

As a possible future work, we could study if other more complex aggregation techniques (i.e. Principal Components (PCA)) could have provided better results. Another possible future work is to analyse whether all the meta-features extracted from the datasets that form the profiles are indeed useful or meaningful for the models to learn how to identify classes. This can be done with feature selection techniques such as Linear Discriminant Analysis (LDA) in order to reduce the profile sizes by only keeping the most significant features.

Finally, the major limitation, specially for the DL approaches, is the lack of available data in the repository. To train the models we only had a dataset of 254 profiles, which is a very low amount of data for the model to learn how the data behaves instead of "memorizing" it.

6.3.1 Research Questions

At the beginning of this thesis, two research questions were formulated. With the results obtained we can argue and answer them.

- ***RQ1: Can Graph Neural Network-based techniques be used for data discovery in Open Data BCN repository?***

As for what has been approached in this thesis, which is the similarity task for data discovery: yes, GNN-based techniques can be used for data discovery in Open Data BCN repository, although, the approach presented in this thesis has not turned out to be very high performing. To sustain this answer we provide the following observations.

The link prediction GCNN model provided in this thesis (detailed architecture in Section 5), has shown an overall good behaviour regarding the classification task it was designed to solve, achieving 100% of accuracy in training and 60.94% in test, which for a 5-class classification problem is considerably good.

One of the reasons explaining why the model does not achieve very good test results is that the way the GCNN is designed, we are asking the model to predict links of a node that it has never seen (which is known as *inductive link prediction* [44]). Whereas in typical link prediction approaches, the model predicts new edges in the same graph it has learnt from in the training phase (this is known as *transductive link prediction*). To understand the limitation caused by this fact we need to analyse what is going on behind the message passing algorithm implemented by the model. In our approach, we obtain a perfect edge prediction in the training stage meaning that we learn a meaningful weight matrix W following notation from Equation 2. This means that after some iterations we learn W such that node features are encoded and mapped to a dimensional space where it is easy to determine whether a pair of nodes contains an edge or not. Following Equation 2, W operates on the adjacency and degree matrices A and D respectively. In our case, when we add a new node to the graph (representing a new dataset from Open Data BCN) to predict its edges, we modify both A and D matrices to include this node which is only linked to itself. In order to find edges for the new node, we forward its initial features (the actual profile of the dataset) to the GCNN defined by the learnt W and aim to obtain a meaningful node embedding in the dimensional space where it is easy to determine what edges should be created from it. However, since W operates on A and D and the new node is

not connected to any other node, the weight matrix only influences the self-loop introduced with the new node obtaining a poor representation of the node embedding that we later compare to the ones obtained in the training phase. On the other hand, if the node from which we want to predict was linked to the right nodes of the graph (following a transductive methodology), the representation we would obtain of the node embedding would be comparable to the ones we obtained in the train phase. Consequently, test result would be more similar to the ones obtained in train. We hypothesise that using an inductive-based approach would lead to better fit of the problem [45, 44].

Still, the fact that we achieve a 60.94% accuracy in the test set with the aforementioned limitation indicates that a considerable amount of the valuable information to determine whether an edge should be created for a given node relies in the node attributes. Then, the lower performance achieved in the test set in comparison with the one obtained in the train one is due to the unknown edge structure of the new node.

Also, the approach proposed may not be that suitable to solve the similarity task of data discovery, but the pipeline provided can be useful to tackle other data discovery tasks such as finding unions or joins. In this thesis, we wanted to approach the "easiest" data discovery task because it was our first experience with graph neural networks, which is why we started with a problem that only required comparisons and classifications at a dataset level. Nonetheless, with some perspective and experience, we realise that working at an attribute level would have helped us to avoid the aggregation problem and would have allow us to work with more data. Also, we could have modeled the input graph from which the model learns in a more sophisticated way leveraging more on the perks of using graph-structured data.

- ***RQ2: Can we notice an improvement in the performance (in terms of accuracy) of GNN approaches in front of more classical data discovery methodologies?***

As for what has been approached in this thesis, which is the similarity task for data discovery: no, we did not obtain better results in the GCNN approach than in any of the other more traditional ML or DL classification approaches that we set as baselines.

The best results obtained are those provided by the Random Forest model, achieving a train accuracy of 100% and a test accuracy of 87.5%. In the GCNN approach we get 100% of accuracy in training and 60.94% in test. In Tables 4 and 6 we can see that in fact, GCNN's are providing good train results, but its test accuracy diminishes considerably. This is probably due to the inductive link prediction problem we stated above. If we had been able to implement the approach in a way that the graph could have learnt properly how to predict edges in an inductive way, the results would likely be improved.

We think, that for this specific task of finding similarities using graph-based deep learning models is not appropriate as with much simpler ML models we obtain more satisfying results. For what we have seen, it is not worth introducing deep learning to solve the classification task for the Open Data BCN repository, as the Random Forest approach provides very good performances.

7 Related Work

In the field of deep learning, graph neural networks are receiving lots of interest recently [46, 25, 47]. These models are showing high capabilities for learning valuable representations from graphs, which are a source of data that is becoming more and more popular every day. Graphs are complex data structures rich in relation information that had hardly been tackled with deep learning. So far, graph neural networks have been applied to recommender systems [48], optimization frameworks [49], study of molecular dynamics [50] and many others. Among all the problems that can be solved with graph neural networks, in this work we focus in link prediction. When tackling link prediction in attributed graphs, neural networks can use both the node attributes and the graph edge structure to add potentially new edges. The standard and most common formulation of link prediction is the *transductive* one [51, 52]. In such framework, graph neural networks predicts new edges between nodes that already have connections with other nodes in the graph. However, many applications require to tackle *inductive* link prediction, where the nodes we are interested in finding edges are unlinked from the graph from which the model learns the edge structure [44]. For instance, in a social network we can leverage graph neural networks to predict new friendship connections for a given user taking into account its existing connections and its profile (attributes), which defines a transductive link prediction problem. However, if we wanted to recommend friendship connections to a new user who just registered we would not have information of any existing connections and therefore we could not exploit any learnt edge structure. For overcoming such limitation, *Hao et.al.*[44] implement two parallel encoders in a single graph neural networks together with an alignment system to rely on attribute information or edge structure information when one of the two is missing. Another very recent paper about inductive link prediction tackles this problem by proposing an efficient way to link the isolated nodes to the rest of the graph by using a K-Nearest Neighbours (KNN) algorithm to transform the inductive problem to a transducting one [45].

Also, in this thesis we used a profiling tool provided by *NextiaJD* system, but their proposal is much more than simply a profiling tool. In [43] they tackle a data discovery task: the problem of discovering joinable datasets at scale. They use profiles to approach the problem in order to avoid unscalable solutions offered by techniques based on index structures like comparison by hash or by value techniques. They also leverage on Spark framework, which offers a distributed system to extract profiles efficiently. They also define a novel notion of join quality, relying on the containment and cardinality proportion, which is capable to detect 5 different quality levels (None, poor, moderate, good and high). NextiaJD uses a predictive model based on a random forest classifier to efficiently rank candidate pairs of joinable attributes using the metric they defined. They run the model over distance vectors obtained by comparing each meta-feature of a normalization of the profiles with min-max and Z-score techniques. In the experiments presented we can see that NextiaJD obtains similar predictive performance than other hash-based methods with a lower amount of false positives and moreover it is able to scale-up to larger volumes of data thanks to the use of Apache parquet formats.

8 Conclusions and Future Work

In this thesis we tackle the similarity task of data discovery in the Open Data BCN repository, which we formalized as classifying its datasets in five thematic areas. To this end, we hypothesized that representing the repository as a graph and using graph-based learning models such as graph neural networks, would be useful to exploit connections between similar datasets providing better results than more traditional methodologies. To validate our hypothesis we provided a Graph Convolutional Neural Network approach solving a link prediction task, that we later mapped into classification, and compared it with other ML and DL baseline methods. The results obtained by our approach are good, proving that it is possible to tackle data discovery tasks with graph-based models, but have not been able to overcome the baseline approaches. Nonetheless, some future work could be able to improve the results obtained by the GCNN making it more likely to outperform some of the baseline models presented. As discussed in Section 6.3.1, in the prediction phase the model is used to predict connections to nodes which are completely isolated from the other nodes of the graph, which is known as *inductive link prediction*. However, the model was trained to solve *transductive link prediction* tasks, where the nodes from which we predict new links are already linked to the graph. To overcome this issue, we should consider implementing one of the state of the art solutions proposed in [44, 45]. This way, the model would be able to find more suitable links for the disconnected nodes and the accuracy would increase.

Although the approach we provided did not turn out to be the best way to find similarities in the Open Data BCN repository, we developed a fully fledged pipeline to solve data discovery tasks in a non-structured and heterogeneous environment, which can be used to solve tasks such as finding joins or unions. Hence, in the context of the project for the city council (i.e., *An Automatic Data Discovery Approach to Enhance Barcelona's Data Ecosystem*), the methodology presented in this thesis can be used to approach the aforementioned tasks which are much more difficult to solve with traditional ML or DL models. The pipeline can also be more useful in the future when the repository will host more public data, as another limitation we encountered was the number of data available to train the models.

As stated in Section 1.1, data integration tasks are often considered the most burdening aspect of data science. Throughout this thesis we had to engage a large data preparation and data cleaning process which consumed a noteworthy amount of our time. It turns out that dealing with Open repositories is not as comfortable as one could think as it is extremely time consuming to find a way to extract the data from it. In this thesis, to overcome this situation, we have developed a scraper which succeeded at efficiently downloading and tagging all datasets from Barcelona's open data website. Hence, as an added value to the project, we render the scraper and the datasets we obtained with it, publishing them in the git repository of the project for all users interested in working with data from Open Data BCN's portal.

Github Repository

All the code generated throughout this thesis can be found at:
`https://github.com/dtim-upc/GCNN-Open-Data-BCN/`. The repository is divided in 4 main folders.

Data

This folder contains the datasets extracted from the Open Data BCN portal after the cleaning step of Section 3.2.1. For storage reasons, Git does not allow to upload files heavier than 25MB, which is why, a few datasets could not be uploaded. The datasets are separated in folders denoting thematic areas exactly in the same way as we can find them in Barcelona's website.

Scripts

This folder contains the scripts we mentioned in Section 3. We have the following:

- 1.scraper.py: Extracts datasets from the Open Data BCN's website.
- 2.rename_datasets.py: Renames datasets to include their class labels.
- 3.clean_datasets.py: Cleans the datasets so they are compatible with the pofiling tool.
- 4.create_information_csv.py: Create a file with necessary information about the datasets in order to create their profiles.
- 5.profiling.scala: Creates profiles of the nominal attributes of the datasets.
- 6.numeric_profiling.py: Creates profiles of the numeric attributes of the datasets.
- 7.create_input_models.py: Prepares training and tests sets and a map of the edges we need to create the input graph for the GCNN method.

Input for Models

This folder hosts the files returned by the code 7.create_input_models.py, which are the ones required to train and test the models. It contains:

- agg_profiles_train.csv & agg_profiles_test.csv: Profiles of the datasets used to train and test the models respectively.
- map_datasets_train.JSON & map_datasets_test.JSON: Dictionary to identify which profile corresponds to which dataset in both the train and the test files.
- map_of_edges_train.JSON: Adjacency matrix of the train set to construct the input graph for the GCNN model.

Models

This folder contains the python notebook hosting the models implemented in Sections 4 and 5. When running them you can obtain the results shown in 6.

- Simpler_models_class_prediction.ipynb: Contains all models from Section 4.
- GCNN_Class_Prediction.ipynb: Contains the GCNN approach explained in Section 5. There is also the final model chosen which provides the results of Section 6.2: model_class_prediction_final.pt

References

[1] Davenport, Thomas H., and D. J. Patil. Data scientist: The sexiest job of the 21st century. *Harvard Business Review 90, no. 10*, (October 2012): 70–76.

[2] Gil Press. Cleaning big data: Most time-consuming, least enjoyable data science task, survey says. *Forbes*, 2016.

[3] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. Data lake management: Challenges and opportunities. *Proc. VLDB Endow. 12, 12*, (2019), 1986–1989.

[4] Renée J. Miller, Fatemeh Nargesian, Erkang Zhu, Christina Christodoulakis, Ken Q. Pu, and Periklis Andritsos. Making open data transparent: Data discovery on open data. *IEEE Data Eng. Bull.*, 41(2):59–70, 2018.

[5] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. Valentine: Evaluating matching techniques for dataset discovery. *arxiv*, 2021.

[6] Miller and R. J. Open data integration. *Proc. VLDB Endow. 11, 12*, page 2130–2139., (2018).

[7] Mary Shacklett. How to tackle data discovery. *The Republic*, 2020.

[8] Val Henson. An analysis of compare-by-hash. In *HotOS*, pages 13–18, 2003.

[9] Dong Deng, Albert Kim, Samuel Madden, and Michael Stonebraker. Silkmoth: An efficient method for finding related sets with maximum matching constraints. *arXiv preprint arXiv:1704.04738*, 2017.

[10] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*, pages 847–864, 2019.

[11] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. Efficient similarity joins for near-duplicate detection. *ACM Transactions on Database Systems (TODS)*, 36(3):1–41, 2011.

[12] Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.

[13] Raul Castro Fernandez, Jisoo Min, Demitri Nava, and Samuel Madden. Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1190–1201. IEEE, 2019.

[14] Yang Yang, Ying Zhang, Wenjie Zhang, and Zengfeng Huang. Gb-kmv: An augmented kmv sketch for approximate containment similarity search. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 458–469. IEEE, 2019.

[15] Erkang Zhu, Fatemeh Nargesian, Ken Q Pu, and Renée J Miller. Lsh ensemble: Internet-scale domain search. *arXiv preprint arXiv:1603.07410*, 2016.

[16] Chen Chen, Behzad Golshan, Alon Y Halevy, Wang-Chiew Tan, and AnHai Doan. Biggorilla: An open-source ecosystem for data preparation and integration. *IEEE Data Eng. Bull.*, 41(2):10–22, 2018.

[17] AnHai Doan, Pedro Domingos, and Alon Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Machine Learning*, 50(3):279–301, 2003.

[18] Mayank Kejriwal and Daniel P Miranker. Semi-supervised instance matching using boosted classifiers. In *European semantic web conference*, pages 388–402. Springer, 2015.

[19] Javier Flores, Sergi Nadal, and Oscar Romero. Towards scalable data discovery. In Yannis Velegrakis, Demetris Zeinalipour-Yazti, Panos K. Chrysanthis, and Francesco Guerra, editors, *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, pages 433–438. OpenProceedings.org, 2021.

[20] Ignacio G Terrizzano, Peter M Schwarz, Mary Roth, and John E Colino. Data wrangling: The challenging yourney from the wild to the lake. In *CIDR*, 2015.

[21] Defining open data. `https://blog.okfn.org/2013/10/03/defining-open-data/`.

[22] Renée J. Miller, Fatemeh Nargesian, Erkang Zhu, Christina Christodoulakis, Ken Q. Pu, and Periklis Andritsos. Making open data transparent: Data discovery on open data. *IEEE Data Eng. Bull.*, 41(2):59–70, 2018.

[23] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[24] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *arXiv preprint arXiv:2005.03675*, 2020.

[25] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[26] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[27] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. Local embeddings for relational data integration. *arXiv preprint arXiv:1909.01120*, 2019.

[28] Raul Castro Fernandez and Samuel Madden. Termite: a system for tunneling through heterogeneous data. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, pages 1–8, 2019.

[29] Christos Koutras, Marios Fragkoulis, Asterios Katsifodimos, and Christoph Lofi. Rema: Graph embeddings-based relational schema matching. In *EDBT/ICDT Workshops*, 2020.

[30] Evgeny Krivosheev, Mattia Atzeni, Katsiaryna Mirylenka, Paolo Scotton, and Fabio Casati. Siamese graph neural networks for data integration. *arXiv preprint arXiv:2001.06543*, 2020.

[31] Jan Salomon Cramer. The origins of logistic regression. 2002.

[32] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[33] Qiang Wu and Ding-Xuan Zhou. Analysis of support vector machine classification. *Journal of Computational Analysis & Applications*, 8(2), 2006.

[34] James M Keller, Michael R Gray, and James A Givens. A fuzzy k-nearest neighbor algorithm. *IEEE transactions on systems, man, and cybernetics*, (4):580–585, 1985.

[35] William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

[36] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017.

[37] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[38] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 2018.

[39] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems (NIPS)*, pages 1–9, 2013.

[40] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.

[41] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International Conference on Machine Learning*, pages 3835–3845. PMLR, 2019.

[42] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th international conference on World Wide Web*, pages 201–210, 2007.

[43] Javier Flores, Sergi Nadal, and Oscar Romero. Effective and scalable data discovery with nextiajd. In Yannis Velegrakis, Demetris Zeinalipour-Yazti, Panos K. Chrysanthis, and Francesco Guerra, editors, *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, pages 690–693. OpenProceedings.org, 2021.

[44] Yu Hao, Xin Cao, Yixiang Fang, Xike Xie, and Sibo Wang. Inductive link prediction for nodes having only attribute information. *arXiv preprint arXiv:2007.08053*, 2020.

[45] Yong-Min Shin, Cong Tran, Won-Yong Shin, and Xin Cao. Edgeless-gnn: Unsupervised inductive edgeless network embedding. *arXiv preprint arXiv:2104.05225*, 2021.

[46] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[47] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[48] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pages 417–426, 2019.

[49] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

[50] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, 2020.

[51] Jinheon Baek, Dong Bok Lee, and Sung Ju Hwang. Learning to extrapolate knowledge: Transductive few-shot out-of-graph link prediction. *arXiv preprint arXiv:2006.06648*, 2020.

[52] Julie Jiang, Li-Ping Liu, and Soha Hassoun. Learning graph representations of biochemical networks and its application to enzymatic link prediction. *Bioinformatics*, 37(6):793–799, 2021.