Network Machine Learning project

Cyrille Pittet (282445) and Alessio Verardo (282634)

May 2023

1 Introduction

One critical aspect of public transport network analysis is the identification of city centers as it is as the beating heart of urban areas. City centers are often characterized by high population densities, commercial activities, cultural landmarks, and transportation hubs. Recognizing these central nodes within a city's public transport network can provide valuable insights into urban dynamics.

This project aims to leverage a dataset of public transport networks from 25 cities around the world to develop models for city center classification. By analyzing the topology, connectivity, and attributes of public transport nodes and edges, we will explore various machine learning techniques to predict whether a specific node belongs to the city center or not.

In the following sections, we will describe the methodology employed, present the dataset used for analysis, discuss the feature engineering and model selection process, and present the results and insights gained from our comparative analysis.

2 Data exploration

2.1 Dataset

The dataset we rely on is [3]. It contains the public transport infrastructure of 25 different cities all around the world. Each stop is represented by a node and two nodes i and j are linked together if there exists a public transports medium that goes from node i to node j. In the following subsection, we explain how we chose to label our data. We also explore the dataset in order to find appropriate features to build our baseline models.

The dataset comes with diverse attributes on both edges (public transport type, average duration, distance between stops, number of vehicles in a given time interval) and nodes (GPS coordinates, name). Note that the list is not exhaustive and the full list can be found in the original paper [3]. As our labelling is based on GPS coordinates, we remove this data from the graph so that the model won't cheat and just learn to classify based on the GPS coordinates.

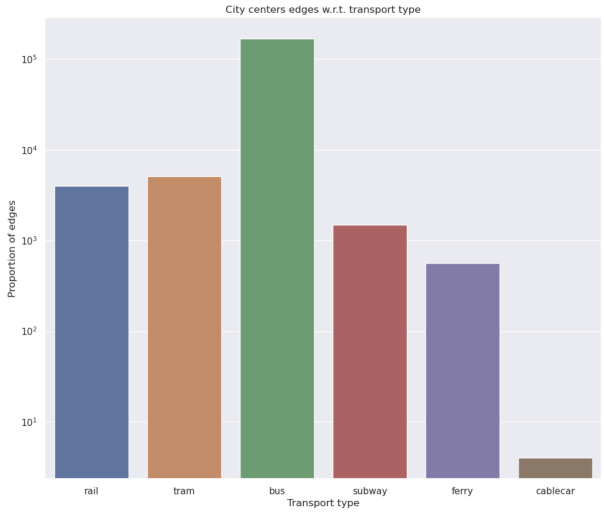


Figure 1: *Public transport type*

2.2 Data labelling

To build a solution to our problem, we had to manually label the stops in the cities as being part of the city center or not. For this manual labelling, we look closely at the different cities in the dataset and saw that for most of the cities, the public transport included in the dataset cover a lot of small towns around the cities, i.e. it is not only the city itself. According to the suggestions of our supervisor, we take the city center and choose a radius for each city such that the city center is included and around 20% of the stops are labelled to be part of the city center (i.e. label 1). As we only have GPS coordinates, we rely on the Haversine distance which is a distance metric used to compute the distance between two coordinates. The specific radius used for each city can be found here.

2.3 Exploration of dataset attributes

In this subsection, we will go through the analysis of the different attributes in the dataset. We start by showing the proportion of each public transport type aggregated over the whole dataset. In fig. 1, we can see that the most dominant transport type is the bus. However, we still have a good proportion of each transport type (except cable car which will be discarded later on). In fig. 2, we show the distribution of transport type for 4 different cities that we hope are good representatives of all other cities in the dataset. We choose two European cities

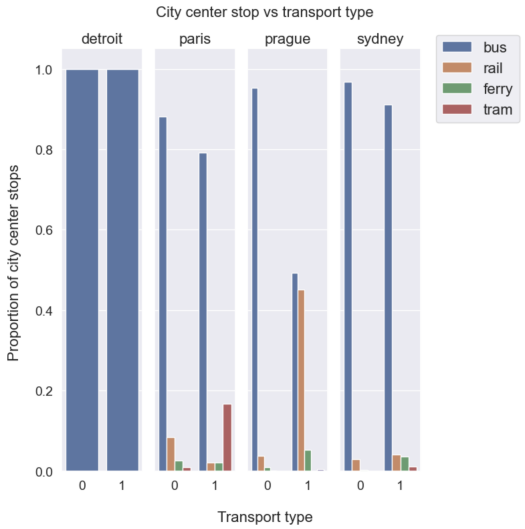


Figure 2: *Public transport type for different cities*

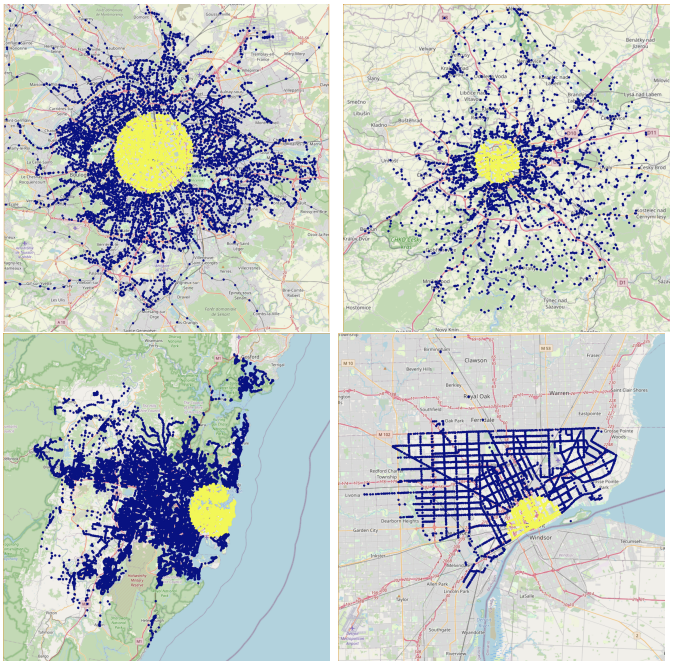


Figure 3: *Illustration of city center for (from left to right, top to bottom) Paris, Prague, Sydney and Detroit. Yellow nodes are nodes in the city center and blue nodes are nodes in suburb.*

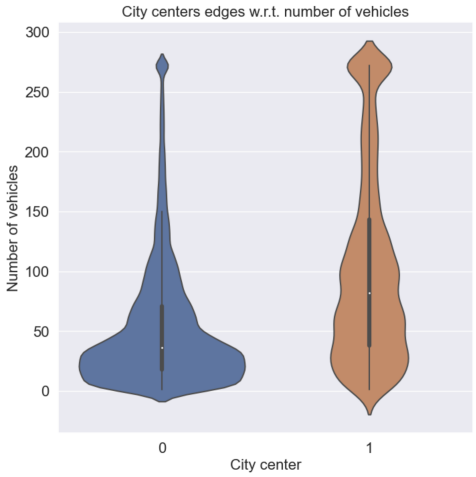


Figure 4: *Distribution of the number of vehicles going through each node for each class*

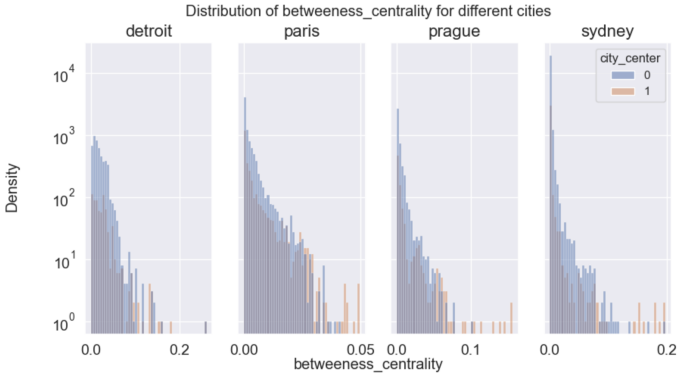
(Paris and Prague), one from the US (Detroit) and one Australian city (Sydney). The cities have different topologies and also different number of nodes. The city center of each city is shown in yellow in fig. 3. We will stick with these four cities as representatives for the whole data exploration section. For the distribution of public transport type, we can see that cities are mostly similar: the bus are the dominant public transport medium. Outside the US, we also have diverse mediums such as trains, ferries and trams. The distribution is also similar between nodes in the city center and not city center. Another interesting feature to look at is the number of vehicles passing on each edge. In fig. 4, we display the distribution of the number of vehicles for both city center and suburb nodes. We can see that the distribution for city center nodes is more biased towards a high number of vehicles than the number of vehicles in suburb. This would be a great feature to discriminate the two categories in our baseline model.

2.4 Exploration of graph properties

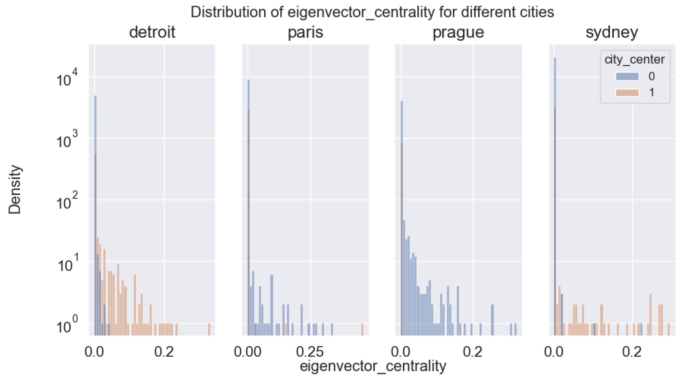
The goal of this section is to explore the graph properties of our 4 cities to determine what are the most important node features to include in our baseline model with handcrafted features. Note that due to the structure of the data, we had to represent the city as a multi-digraph as there could be multiple edges between the same stops. For measures that cannot be computed by the NetworkX library on multi-digraph, we cast the graph as a digraph where multiple edges are represented as one with weight 1. Note that a more complete analysis has been made in the EDA notebook and our first finding in the notebook was that degree distribution are similar for city center and suburb and was never bigger than 15.

2.4.1 Betweenness centrality

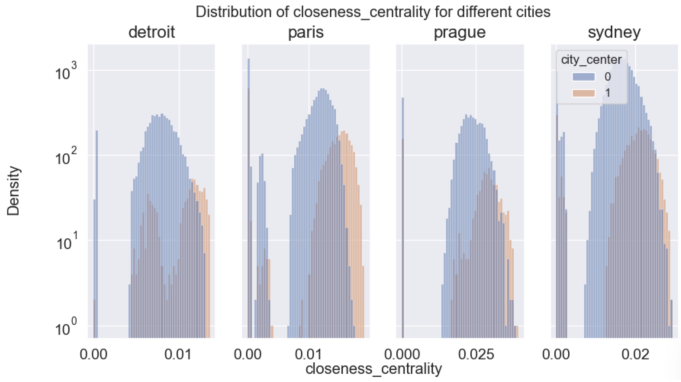
Betweenness centrality is a widely used measure that captures a node's role to pass information from one side of



(a) *Distribution of betweenness centrality for our 4 cities. City center (orange), suburb (blue)*



(b) *Distribution of eigenvector centrality for our 4 cities. City center (orange), suburb (blue)*



(c) *Distribution of closeness centrality for our 4 cities. City center (orange), suburb (blue)*

the network to the other one. Specifically, it is defined as the ratio of shortest paths between all the other nodes going through the node of interest. In essence, it captures how central the node is to go from one side of the network to the other. Applied to our dataset, we could think that node in the city center would have a high betweenness centrality. However, as shown in fig. 5a, it is not as clear as we thought. The main reason is that our city center nodes are not always at the center of the graph. For instance, in Detroit, the city center is located near an edge of the city (fig. 3). However, there is still a tendency to have betweenness centrality distribution with a heavier tail for the city center nodes.

2.4.2 Eigenvector centrality

The eigenvector is a relative score that is assigned to all nodes in the network. It is based on the same concept as PageRank where connections to high-scoring nodes contribute more to the score of the current node than having connections with low-score nodes.

It is interesting to note that, in this dataset, there is a clear difference between the two distributions but not always in the same direction. For city that are formed as a ring (like Paris and Prague), the eigenvector centrality of city center nodes will be lower than suburb. However, for other city like Detroit and Sydney, the distribution is inverted and eigenvector centrality is much higher for city center than suburb. This comes from the geographical position of our defined city center in the city as shown in fig. 3. In cities like Sydney and Detroit, the city center is at an edge of the graph whereas Prague and Paris have a city center which is at the center of the graph.

Therefore, this feature will be nice to discriminate between the city center and suburb within a city, e.g. if we want to train on parts of the network and predict new nodes. But it won't be sufficient to work on an unseen city as it seems to be dependent on the structure of the graph.

2.4.3 Closeness centrality

Closeness centrality is calculated as the reciprocal of the sum of the length of the shortest paths between the node and all other nodes in the graph. Thus, the more central a node is, the closer it is to all other nodes. We can see in fig. 5c that there is a clear shift between city center and suburb. That is, the closeness centrality distributions seems to be always a bit more to the right for city center nodes. More importantly, it is the case for all cities (not like the eigenvector centrality) and means that we have here a good features to discriminate city center and suburb for all cities.

2.4.4 Graph models

If we think about the type of graph we are dealing with, we can easily see that it doesn't fit any of the network models we saw during the lectures.

- Erdös-Rényi: our graph is clearly not a random graph as all connections between nodes are well thought by humans in order to optimize the public transport infrastructure.
- Barábasi-Albert (Scale free): From the degree distributions plots available in the notebook, we can see that they do not follow a power law. Since this property is not met, it cannot be a BA-graph.
- Watts-Strogatz: During the lectures, we saw that this type of networks should meet two properties: a small average shortest paths and a high clustering coefficient. We can already see that the small-world property is not met as two nodes that are far away in the graph will need to go through a lot of nodes to reach each other, i.e. there are no shortcuts from one side of the network to the other. The clustering coefficient is likely to be low as neighbors of a stop are not likely to be connected together as this won't be efficient in a public transport infrastructure. This observation is verified in the Exploratory Data Analysis notebook.

With all these explanations, we can see that we won't be able to use any of the properties derived in the lecture directly.

3 Exploitation

In this section, we will describe the different methods we used to perform our classification task. To start, our problem can be split into 2 different tasks:

- Train to predict the label of a subset of nodes in one city and test the performances on another subset of the same city (intra-city training)
- Train to predict the labels of node on a subset of cities and predict the labels on other cities (inter-city training).

The second option will allow us to test how well our models generalize to unseen cities.

3.1 Baseline model

For our baseline models, we try two different approaches. The first one is to use handcrafted features spotted during the exploration part. These features will then be fed to a SVM classifier to predict if a node is part of the city center or not. This approach allows us to use different attributes included in the dataset. The second approach is to work only with the structure of the graph and use Node2Vec to derive node embeddings and fed them to a SVM classifier. Note that testing the generalization of the models on unseen cities is only possible using the first solution. To derive node embeddings with Node2Vec, we need to do random walks in the graph. Since cities are disconnected, we cannot have consistent embeddings for all the cities.

3.1.1 Handcrafted features

We extract the following features as handcrafted features:

- Duration average from incoming (resp. outgoing) traffic.
- Number of vehicles in incoming (resp. outgoing) traffic.
- Betweenness, closeness, Katz and eigenvector centralities.
- (Normalized) Histogram of traffic, e.g. number of traffic of each type (bus, tram, rail, subway, ferry).
- Distance of incoming (resp. outgoing) traffic.
- In and out-degree distribution.

In the result section, we will see what performances these handcrafted features can achieve and also which of the features are the most important. We will also explore the ability of generalization of these handcrafted features to unseen cities.

3.1.2 Node2Vec

A second way of creating a baseline is to rely on the Node2Vec algorithm. Introduced in [2], this algorithm starts by generating different random walks from the starting node. These random walks are driven by two parameters p and q. Intuitively, the larger q is, the more we force the random walk to look like a DFS (as opposed to a BFS with a small q). The larger p is, the more we force the random walk to take a step back and to return to the previous node. These random walks are then fed to a Word2Vec [4] algorithm aiming at creating embeddings such that nodes that appear near to each other have a similar embeddings to these other nodes. Originally, Word2Vec [4] was designed for Natural Language Processing, represent sentences as a sequence of tokens (i.e. numbers) and want to generate embeddings that are close (w.r.t. the cosine similarity) for words that appears together in sentences. For the graph model, the random walks, i.e. sequence of nodes, obtained in the algorithm act as the sentences, i.e. sequence of tokens, that are fed to the Word2Vec algorithm.

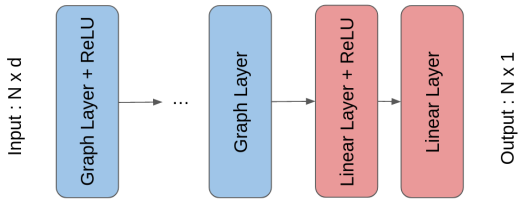


Figure 6: *Generic architecture of the GNN. N is the number of nodes in the input graph, d the dimensions of the node features.*

3.2 Graph Neural network model

In a second stage, we rely on graph neural networks (GNN) to perform the task. Two architectures are considered : one using the graph convolution operator from [5] and one using the graph attention (v2) operator from [1]. The generic architecture is shown in fig. 6 and they are detailed in the corresponding sections below (3.2.1, 3.2.2). It consists of multiple graph convolution layers, graph attention layers, respectively, followed by two linear layers to do the classification.

From the baseline models experiments, we observe that the handcrafted features yield better performances. Therefore, we will only use these features to train the GNN models. Another reason for this is that the Node2Vec features only encode topological information to the contrary of the handcrafted features that bring new information. However, the GNNs already take into account this topology thanks to their architectures, in contrast to a SVM model. Therefore, using the handcrafted features with the GNNs allows to mix both source of information.

To split the data into training, validation and test sets, we separate the nodes of a city randomly into 3 sets, while keeping the same proportions in the labels. During the training, the features of all the nodes are fed to the model and they are all classified. Then, we only pass the outputs corresponding to the training nodes to the loss function. This has the effect to use the representations of all the nodes to compute the representation of each node but the weights are updated only based on the outputs of the training nodes.

3.2.1 Graph Convolution Layer

We use the graph convolution operator from [5]. At each layer, the features of node i are updated as follows :

$$x'_i = W_1 x_i + W_2 \sum_{j \in N(i)} e_{ji} x_j \tag{1}$$

where x_i is the feature vector of node i, e_{ji} is the weight of edge $j \to i$, $N(i)$ are the neighbors of node i, W_1 and W_2 are learnable parameters.

We can control the number of such layers we use in the model as well as the dimensionality of the feature vectors x'_i.

City	Node2Vec	Hand	GNN	GAT
Adelaide	46.78	57.82	**58.27**	47.59
Belfast	43.98	**56.97**	47.27	37.50
Berlin	69.41	56.15	68.90	**71.69**
Bordeaux	35.56	**68.61**	67.56	44.60
Brisbane	52.37	**56.08**	54.56	51.93
Canberra	49.22	43.15	44.44	**54.63**
Detroit	52.09	54.55	59.54	**64.41**
Dublin	51.36	59.88	52.07	**60.96**
Grenoble	39.42	**51.75**	46.29	49.01
Helsinki	**81.86**	63.23	75.66	81.14
Kuopio	43.48	**47.62**	30.76	25.64
Lisbon	**58.67**	55.73	58.31	53.52
Luxembourg	35.20	**45.61**	40.38	42.00
Melbourne	**61.80**	54.93	52.39	57.07
Nantes	39.58	**56.25**	49.24	43.78
Palermo	29.36	**71.77**	59.78	42.42
Paris	70.21	70.83	74.12	**74.77**
Prague	62.41	**68.09**	64.08	**68.01**
Rennes	63.45	67.59	**72.72**	70.07
Rome	41.67	**65.05**	64.95	54.37
Sydney	37.23	**50.12**	49.46	47.20
Toulouse	40.16	54.15	**56.88**	53.26
Turku	28.85	**65.50**	63.44	34.90
Venice	58.49	**68.57**	64.00	66.66
Winnipeg	36.16	**51.55**	45.02	44.82

Table 1: *F1-score for class 1 over the test set of each city.*

3.2.2 Graph Attention Layer

For the graph attention layer, we use the GATv2 proposed in [1]. It has been shown to outperform the original graph attention layer on different benchmarks. In this layer, attention coefficients are first computed :

$$\alpha_{ij} = \frac{\exp\left(a^T \text{LeakyReLU}(\Theta[x_i || x_j])\right)}{\sum_{k \in N(i) \cup i} \exp\left(a^T \text{LeakyReLU}(\Theta[x_i || x_k])\right)} \tag{2}$$

where Θ are the learnable parameters, $[x_i || x_j]$ is the concatenation of vectors x_i and x_j, $N(i)$ are the neighbors of node i.

Then the node features are updated :

$$x'_i = \alpha_{ii} \Theta x_i + \sum_{j \in N(i)} \alpha_{ji} \Theta x_j \tag{3}$$

In this layer, we control the number of layers we use as well as the dimensionality of the node features. Similar to the transformer architecture in NLP, we can also use multiple attention heads. This simply applies multiple parallel updates as shown above, each with an independent set of parameters Θ. The outputs of these heads are then concatenated into a single feature vector for each node.

4 Results

4.1 Handcrafted features

Interestingly, it seems that our handcrafted features perform really well as we obtained an average f1-score of 58.46% compared to 49.15% for the Node2Vec based method. It seems that the attributes on the edges that we used to create the features are more helpful than the structure of the graph for classifying the nodes as being part of the city center or not. We will now take a deeper look at which features are the most useful.

For this purpose, we rely on an F-test for classification. It computes how good is a feature to discriminate two classes based on the residual sum of squares. Specifically, we compute the following F-test for a given feature X

$$F = \frac{\frac{SSm - SSf}{p_{fit} - p_{mean}}}{\frac{SSf}{n - p_{fit}}}$$

where

- $SSm = \sum_i (X_i - \mu)^2$ with $\mu = \frac{1}{n}\sum_i X_i$
- $SSf = \sum_{j=0}^{1} \sum_{i \in C_j} (X_i - \mu_j)^2$, C_j represents all the data points from class j and $\mu_j = \frac{1}{|C_j|}\sum_{i \in C_j} X_i$.
- $p_{fit} = (n_0 - 1) + (n_1 - 1)$ where $n_i, i \in \{0, 1\}$ represents the number of samples of class i.
- $p_{mean} = nb_c - 1$ where nb_c is the number of classes and is 2 in our case.

The F-value represents the ratio of the between-class variance to the within-class variance and is used to assess the significance of each feature's contribution to the target variable. A high F-value indicates that the feature has a strong influence on the target, while a low F-value suggests that the feature may not be informative for classification. To test how important is each features, we compute the p-values on each city and we report the 10th, 50th and 90th percentile of p-values in table 2. In table 2, we can see that some of the percentiles are NAN. This is because the corresponding features are constant across all stops in one city. This happens, for example, in Detroit where we have only buses in the public transport infrastructure, therefore the histogram of public transport medium will be constant for the city. Another example is that most of the cities don't have any ferry because they are not necessarily close to the sea.

From this table, we can also see that 90% of the time, the features n_vehicles_out, n_vehicles_in and closeness centrality are statistically significant (i.e. they have a non-negligible importance at reducing the variance of the features when we group them by classes) at level 10^{-4}. Another feature that seems very useful most of the time is the betweenness centrality. If we were to reduce the number of features, e.g. to do a faster system, we would remove the normalized histogram of transport, eigenvector centrality, d_in, d_out based on table 2 which already reduces the features dimension by 7. We tried to do this but it decreases the performance by 4% of F1-score so although they are not the most useful features, they are still useful to increase the performance of the model.

Feature	**10th**	**50th**	**90th**
in_degree_distribution	5.93E-17	7.81E-05	7.61E-01
out_degree_distribution	1.97E-16	5.25E-06	6.47E-01
clustering	1.40E-06	4.79E-02	6.61E-01
betweenness_centrality	1.03E-31	2.86E-14	1.78E-01
eigenvector_centrality	4.99E-34	9.37E-03	5.64E-01
katz_centrality	3.41E-20	8.41E-08	5.17E-01
closeness_centrality	1.46E-134	2.64E-52	5.80E-04
tram	7.68E-65	1.31E-01	NAN
subway	1.05E-11	NAN	NAN
rail	5.18E-03	NAN	NAN
bus	2.17E-110	NAN	NAN
ferry	3.20E-06	NAN	NAN
d_in	5.00E-07	2.18E-03	3.71E-01
n_vehicles_in	1.34E-227	1.93E-81	9.31E-14
duration_avg_in	1.18E-23	9.88E-08	2.85E-01
d_out	3.92E-07	2.05E-02	3.45E-01
n_vehicles_out	3.16E-231	1.15E-82	1.10E-11
duration_avg_out	1.85E-22	6.51E-08	4.30E-01

Table 2: *Different percentiles of p-values for each feature across the cities in the dataset.*

Split	**F1-score**
Train	52.16
Test	52.81

Table 3: *F1-score for class 1 over the test set of 5 cities.*

Finally, we also take a look at the generalization ability of the handcrafted features. Specifically, we choose 20 cities at random and we train to predict the label of each node on them. Then, our test set, composed of 5 different cities, is used to assess the generalization performance of the model displayed in table 3. As we can see, the performance are not so good as we lose at least 10% of F1-score with respect to the best model for each city. This may be due to the fact the cities are really different between each other and some of the features are super-good for one city but bad for another one. Also, as the train f1-score is smaller than the validation one, we can also see that the model didn't perfectly fit the data.

4.2 Node2Vec

We started our exploration here by looking at the values of p and q that would suit our problem the best. As explained before, p controls how much we go back to the previous node and q controls if the random should look like a DFS (q large) or a BFS (q small). For our problem,

we think that q should be small so that we are more biased towards our neighborhood rather than nodes that are far away. This intuition comes from the fact that our neighborhood should define our starting point as nodes close to each other are likely to have the same label. Also, as all nodes that belong to the city center are grouped together, having more information about the neighbors seems more useful than nodes that are far away in the network. For p, we didn't really know a priori what should be the value, i.e. it should have a medium value (not too large and not too low).

To determine what are the best values for the parameters p and q, we perform a cross-validation where we took only 4 cities of different shapes and number of nodes (Adelaide, Detroit, Paris, Turku). We train one model for each pair of p and q in the ranges $p \in [0.1, 1.0, 5.0]$ and $q \in [0.1, 0.5, 1.0, 2.0, 5.0]$ and for each city. We take the pair of parameters that performs the best with respect to the average F1-score (for class 1) over the 4 cities over the validation set.

Detailed results can be found in the following file. The best was given by $p = 5.0$ and $q = 0.1$, dimension of embeddings 256 and 100 random walks from each node. We can see that our intuition was therefore correct. In table 1, we can see that Node2Vec didn't really perform well even with the best pair of parameters. Most of the time, it performs on par or lower than the handcrafted features. These results were surprising at first as we expected the Node2Vec based models to perform better. But the more we thought about it and the more it makes sense. First of all, our graph is structured in "grid fashion", i.e. most of the time neighbors of a node are not connected together and the node in the city center are connected the same way as in the suburb areas. Therefore, since Node2Vec uses only the structure of the graph to generate embeddings (and not any of the attributes of the nodes), it is likely that the embeddings won't convey enough information.

4.3 Graph neural network

Similar to what is done in 4.2, we started by searching for good hyperparameters. For both models, these are the number of layers and the dimensionality of the feature vector. For the graph attention model, we also have the number of heads. The cross-validation is done the same way as in 4.2.

We use the AdamW optimizer with a learning rate of 10^{-3}, the binary cross-entropy (BCE) loss and train the models for 200 epochs. The model is tested on a validation set after each epoch and the model with the best validation F1-score is kept. Since the training set is not balanced in terms of labels, we reweight the samples with label 1 (`pos_weight` in PyTorch) in the loss with the ratio of the number of samples with label 0 over the number of samples with label 0.

Since adding a graph layer (convolution or attention) allows to reach 1 more hop in the graph, we think that we should keep the number of layers rather low to avoid averaging out the information from neighboring nodes. For the graph convolution, we tried $n \in \{3, 5, 10\}$ and $n \in \{3, 5, 10\}$ for the graph attention. For the dimensionality of the features, we used $d \in \{16, 32, 64, 128\}$ for the graph convolution. For the graph attention, it is slightly different as the output of the layer has dimensionality $h \cdot d$ where h is the number of heads and d the dimensionality of a single feature vector. Therefore we chose $d \in \{8, 16, 32\}$ and $h \in \{2, 4, 8\}$ to have an output dimension similar to the graph convolution.

The detailed results can be found in this file. For the graph convolution model, the best values are $n = 3$, $d = 32$. For the graph attention model, it is $n = 3$, $d = 32$, $h = 8$.

In table 1, the observation is that the two neural network do not constantly outperform the baseline models and their performances can be quite different from one city to another. One of the neural network performs best in only approx. one third of the cities and two thirds of these cases, the graph attention model performs best. A first possible explanation for these differences is the size of the city. If the size of the city is rather low, as Kupio, then the model (which has a fixed size for all cities) might overfit more easily compared to larger cities such as Paris. In a sense having more nodes acts as a regularization technique. This also suggests that having a single fixed model with fixed hyperparameters might not be well suited for this task and using personalized models might be beneficial to harness the full potential of neural networks.

5 Conclusion

In this project, we have explored the task of predicting city center nodes in public transport networks across 25 diverse cities worldwide. We manage to go through the whole data science pipeline: exploration, labeling and exploitation. We identified handcrafted node features that are relevant to the task. We also learnt unsupervised node features with Node2Vec in order to train a baseline SVM classifier. We first considered model personalized to each city and observed that the performances are not consistent throughout these cities. Also the best approach to compute the features (handcrafted vs Node2Vec) seems to depend on the particularities of the topological structure. This suggests that the using a global model train on a subset of cities and test on unseen cities is a hard task. Indeed, we observed that the results of generalization were lower than the personalized models. We then considered two architectures of Graph Neural Networks which takes as input our handcrafted features. Suprisingly, the GNNs do not always bring a significant increase of performance, even the opposite for some cities. We think that it might partially be due to the difference in size of the cities which

might lead to easier overfitting in some cases. This suggests that a fixed architecture, regardless of the city particularities, is not appropriate and we may benefit from using tailored architectures.

References

[1] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2022.

[2] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016.

[3] Rainer Kujala, Christoffer Weckström, Richard K. Darst, Miloš N Mladenović, and Jari Saramäki. A collection of public transport network data sets for 25 cities, May 2018.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[5] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks, 2021.