Road Network Analysis in Berlin, Germany

Bill Sendewicz

Hannula-Katrin Pandis

Canberk Özen

Abstract—**Traffic networks are ubiquitous in daily life, as is the congestion of road networks. One way to approximate the congestion of a road network is by using the actual travel speed of trips made in the network. We analyzed the mean travel speed of Uber trips made in Berlin, Germany between 7 AM and 10 AM in the first quarter of 2020 to better understand traffic congestion in that city. At first we tried to model traffic congestion directly, and in order to do this, we combined two different road network datasets. However, after combining the two datasets, we ended up with a very sparse dataset that rendered our initial plan invalid. So we pivoted and instead predicted the mean speeds of the roads since that can be used as a proxy for traffic congestion. Linear regression, random forest regression and Node2Vec models were employed in a supervised learning setting to predict mean speeds in Berlin. The feature sets we used were the road attributes, road attributes combined with simple unsupervised features extracted from the graph, and the node embeddings returned by Node2Vec. The lowest MSE score was achieved with the road attributes combined with simple unsupervised features extracted from the graph together with random forest regression.**

Keywords—traffic networks, OpenStreetMap, OSMnx, linear regression, random forest regression, Node2Vec

I. Introduction

Traffic congestion on urban road networks has increased substantially since the 1950s. When traffic is great enough that the interaction between vehicles slows the speed of the traffic stream, this results in road congestion. Congestion can be predicted by a speed metric in a road network, as it is a linear multiple of the mean speed of the road. This project focuses on analyzing the mean speed of the cars in the road network in Berlin, Germany.

Traffic congestion is costly both in terms of economics and in terms of human health. Predicting it accurately will help municipalities with urban planning and may even help social scientists gain new insights into their work.

Our motivation for choosing a project related to traffic data came from our interests—we all have had some previous exposure with traffic data and we wanted to gain more practical experience working with this type of dataset. Moreover, we also found that traffic datasets are easily understandable as traffic of one form or another is something we all experience on a daily basis.

The dataset we analyzed is the network of roads meant for driving in Berlin from OpenStreetMap. It contains 27,956 nodes and 72,908 edges in total, whereas the nodes represent intersections and edges represent roads connecting two intersections.

The graph we got from OpenStreetMap has a variety of edge attributes, such as road type, maximum speed of the road, whether the street is one- or two-way and its length, with the maximum speed attribute being the most important for this particular project. The maximum speed does not represent the actual travel speeds on the roads; it represents the legal speed limit of the road.

We enriched our OpenStreetMap data with the movement dataset provided by Uber from Q1 in 2020. The nodes represent the intersections of roads and edges represent roads, the same as in OpenStreetMap. An edge is present in Uber's dataset if there were at least five rides connecting two nodes. After merging the Uber and OpenStreetMap datasets, the number of edges decreased from 73,000 to 9,000. The Uber dataset provides the average speed, standard deviation of speed, median speed, and 85th percentile speed by hour of day, aggregated from all days in the quarter. All the speed metrics are in kilometers per hour, as the maximum speed in OpenStreetMap dataset. The Uber dataset has also an hour of the day feature, and for the project scope we chose the data between 7 AM to 10 AM and treated this as one aggregate timeslot. We preprocessed our data and left out features that did not give us any value, such as bridge, tunnel, access and lane indicators.

We created a baseline model and two additional models for the mean speed prediction. The baseline model was created by predicting mean travel speeds on OpenStreetMap merged with Uber dataset—the so-called vanilla dataset. The dataset was split into two parts, 75% was for the train set and 25% was for the test set, and used supervised learning methods. Linear regression and random forest regression methods were used for the prediction.

The first approach was to enrich the travel speed dataset with network-specific features and see if and how these improve the mean travel speed predictions. Features that we used were Common Neighbors, Jaccard Similarity, Adamic Adar, Katz score and Preferential attachment. The graph features we calculated were fed into linear regression and random forest regression models.

The second approach was to predict travel speeds by using features returned by Node2Vec. For Node2Vec, we used the same dataframe as for the baseline models, meaning the graph features were not used. The first model was created from our preprocessed dataset using all columns as edge attributes. The second model was using only mean speed as an edge attribute and the third model was otherwise the same as the second, but had slightly different Node2Vec parameters. Features we received were fed into linear regression and random forest regression models.

For evaluation, the mean squared error (MSE) was used. Our best result was using a random forest regression model on the vanilla dataset augmented with network-specific features, with an MSE of 64.29.

II. Related Works

Link prediction is the key problem for network-structured data. Score functions, like common neighbors and Katz index, are used to measure the likelihood of the links and therefore the link prediction. These are easy to use and interpret, yet these have strong assumptions on when two nodes are likely to link, which can become limiting if the assumptions fail.

It would be more effective to use a local subgraph from the network to learn the function mapping the subgraph patterns to link existence, thus to learn a heuristic that suits the network. The extracted local enclosing subgraph around the links can be used as training data and using a fully-connected neural network (NN) to learn which subgraphs correspond to link existence. This can be used as a state-of-the-art tool for link prediction.

Reference [1] proposes learning heuristics from local subgraphs using a graph neural network (GNN). GNNs, instead of fully-connected NNs, will enable better graph feature learning ability and permit learning from not only subgraph structures, but also latent and explicit node features, thus absorbing multiple types of information.

Reference [2] summarizes progress made in 2010 about link prediction algorithms, emphasizing the random-walk-based methods and the maximum likelihood methods.

The simplest framework of link prediction methods is the similarity-based algorithm, where each pair of nodes is assigned a score, which is directly defined as the similarity between the nodes. Some examples are Common Neighbors, Jaccard Index, Resource Allocation Index, Katz score.

Maximum Likelihood Methods, like Hierarchical Structure Model, Stochastic Block Model, presuppose organizing principles of the network structure, with the detailed rules and specific parameters obtained by maximizing the likelihood of the observed structure. These are time consuming and probably not the most accurate, though these provide valuable insights which cannot be gained from similarity-based algorithms.

Based on [2], most of the studies for link prediction were focusing on unweighted and undirected networks up until 2010.

Reference [3] proposes a new deep learning framework called a Spatio-Temporal Graph Convolutional Network (STGCN) that improves the shortcomings in traffic forecasting of metrics of traffic flow, such as speed, volume and density. The authors classify the length of prediction into two classes, short-term (5-30 minutes) and mid-and-long term (30+ minutes). Statistical methods such as linear regression work well on short-term forecasts, but not for mid-and-long term forecasts. For the latter class, dynamical modeling and data-driven approaches are typically used. Dynamical modeling requires massive computational resources and simplified assumptions that reduce prediction accuracy. So the authors proposed a deep learning architecture that models temporal dynamics as well as spatial dependencies of traffic flow. Their experiments show that their STGCN outperforms other state-of-the-art data-driven prediction frameworks with faster training, easier convergence and fewer parameters, while achieving flexibility and scalability.

Reference [4] asserts that graph convolutional networks are not adequate to perform machine learning tasks on road networks, such as traffic forecasting, speed limit annotation or travel time estimation, for three reasons: many implicit assumptions in GCNs do not hold in road networks and even the best GCNs can only consider one source of attribute information at a time, such as node or edge attributes. Secondly, road networks are very low density. For example, the Danish road network they studied has an average node degree of 2.2, whereas citation networks can have an average node degree of around 10 and social networks can have a node degree of 500 or more. Third, GCNs assume that the network exhibits homophily, i.e., that nearby nodes or edges are similar to each other. This is not always the case with road networks, such as when speed limits change drastically between two neighboring roads. To overcome these shortcomings, the authors propose what they call a Relational Fusion Network (RFN) that is based on a relational fusion operator--a type of novel graph convolution operator--that aggregates over representations of relations rather than representations of neighbors. RFNs can consider the relationships between nodes (road intersections) and edges (road segments) jointly using a node-relational and edge-relational view of the network. Their RFN model outperforms GCN baselines by at least 32% on driving speed estimation tasks and at least 21% on speed limit classification tasks.

Reference [5] introduces Relational-GCNs, which are based on graph convolutional networks but adapted to knowledge base completion tasks. Their model aims to solve two important problems, namely entity classification and link prediction. For the former, R-GCN works similarly to a GCN in the sense that it uses softmax classifiers at each node. The R-GCN network feeds the node representations to those classifiers which then predict the labels. The latter, on the other hand, is more complex to solve. In link prediction tasks, the R-GCN works as an encoder, which produces latent feature representations of entities; and a decoder, which is a tensor factorization model (they used DistMult, specifically) that then predicts the edge labels using the encoded representations.

The main importance of [5] is the application of GCN architecture on modeling relational data, and more specifically, R-GCNs' ability to scale into knowledge graphs containing a large number of different types of relations via parameter sharing between the weight matrices of such different triplets. The authors also show that the performance of factorization models like DistMult can be improved if they are enriched with an encoder model (their encoder enriched model contributed 29.8% improvement on the FB15k-237 dataset).

Similar to R-GCNs, the MagNet model in [6] also aims to solve node classification and link prediction tasks on knowledge graphs. The difference is in its approach: it is a spectral network unlike the R-GCN (which is a spatial network). To this end, the authors use a different kind of Laplacian, called the magnetic Laplacian, which is able to encode the asymmetric nature of a directed graph vis-à-vis a graph Laplacian defined with real-values only.

Specifically, the Magnet model consists of two spectral graph convolutional layers (created with the help of complex-valued magnetic Laplacian) plus a complex ReLU. After the input features are processed in these layers, the real and imaginary parts of the outputs are separated. These two feature matrices are then fed into a fully connected layer followed by softmax.

In terms of link prediction accuracy, MagNet took the lead in seven out of nine datasets compared to the performance of several models like Chebyshev Nets, GATs, spectral GCNs (which are designed for undirected graphs).

III. Dataset

We selected two datasets for this project and combined them to create the final dataset we used for analysis. More specifically, we downloaded the vehicle road network data of the city of Berlin from OpenStreetMap (OSM) through the OSMnx library. In addition, we also used the movement data of Berlin provided by Uber, which recorded the information of Uber rides conducted in Berlin.

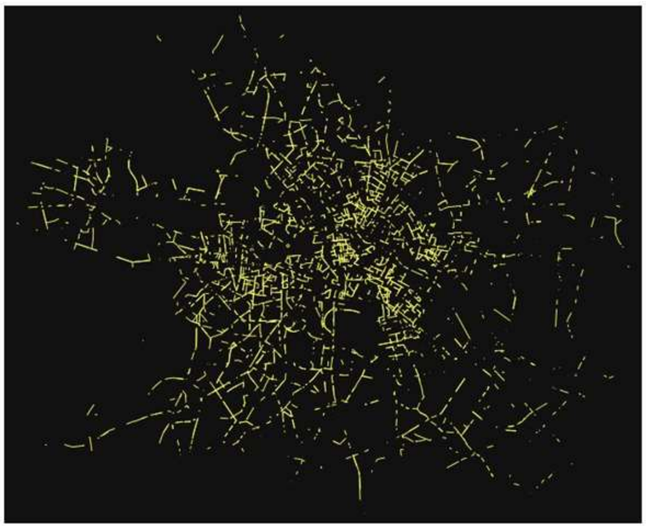
The main addition from the Uber dataset is the mean speed of road segments by hour of the day. We decided to fetch the movement data existing in the timeframe between 7 AM and 10 AM from the first quarter of 2020. An edge is present in Uber's dataset if there were at least five rides connecting two nodes.

For creating the final dataset, we used the shared node IDs between the two datasets and merged them into a single dataset. After merging them, we converted our multi-directed graph into a directed graph. The final dataframes for edges and nodes included several features, with edges having significantly more features than the nodes. Important features include road length, road type and the speed limit allowed for the road.

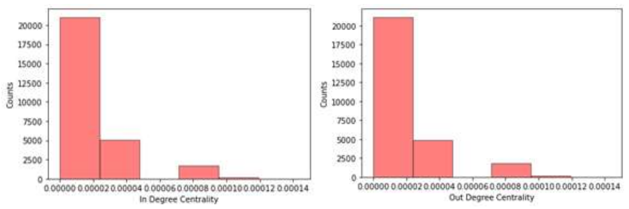
One particular thing to note here is that the original OSM data for Berlin had 72,908 edges and 27,956 nodes, but after merging our datasets, our edge count was reduced to 9012. This resulted in a sparse network, with node counts being much larger than that of the edges.

The road network of Berlin provided by OSM before merging can be seem below:

After merging the network above with the Uber Movement data, we ended up with a much sparser graph as seen below:



Before discussing the methods we applied, it is worth noting the in-degree and out-degree centrality distributions of our nodes. The histograms look very similar, which suggests that most of our nodes are connected in a bidirectional manner, that is, most streets in Berlin are two-way streets.



IV. Methodology

Our main aim for this project was to come up with a model for traffic congestion in a certain period. But after we ended up with a very sparse road network, we decided to focus on predicting the mean speed attributes of the roads, since modeling traffic congestion would not be worth the effort on such a sparse dataset with too many roads missing in the graph.

To predict the edge attribute, the mean speed of the road, we used a linear regression and a random forest regression model with 100 trees to see if the non-linear model would be able to capture the patterns in our data better than the linear one. In addition, we used three different sets of features, incrementing their complexity at every step. We'll now describe them in detail.

Baseline Features

The so-called vanilla dataset included the following features: source node, target node, highway (road) type, max speed, one-way indicator (boolean), road length and hour of day. The target variable is mean speed in kilometers per hour. We used this labeled dataset to feed into linear regression and random forest regression models to predict the mean speed and then compared the predicted labels against the actual labels.

Baseline Features + Unsupervised Features Extracted according to the Graph Structure

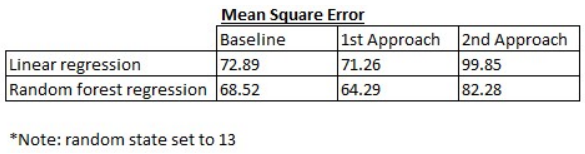
The second approach is based on the first one, with the same dataset used to generate calculated graph features. Unsupervised measurements rely on different structural properties of networks. We used neighborhood measures like Common Neighbors, Adamic Adar and Jaccard Similarity in this project. In addition, we used path-based measures like Katz score and vertex feature aggregation like preferential attachment. Calculated measures were added to the existing dataframe and the mean speed attribute was predicted using a linear regression model as well as random forest regression model.

Node2Vec

Besides the two relatively basic set of features above, we also applied the Node2Vec algorithm to our graph and used the embeddings returned by it as our features for both of our models. All of the embeddings had a dimension size of 30, a walk length of 7 and the number of walks equal to 50.

We tried several combinations for return and in-out parameters ($p = 10$, $q = 0.25$ and $p = 1$, $q = 10$), emphasizing breadth-first search and depth-first search separately. In addition, we used Node2Vec both with edge weights and without, with edge weight defined as our target feature, the mean speed of the road.

V. Results



Mean Square Error

	Baseline	1st Approach	2nd Approach
Linear regression	72.89	71.26	99.85
Random forest regression	68.52	64.29	82.28

*Note: random state set to 13

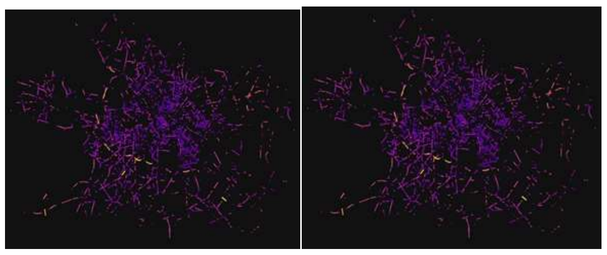
The results we achieved are displayed above with the three methods mentioned. We used a 75% train 25% test split when modeling. The best results on our test set were achieved with a random forest regression model combining the baseline features such as highway type and road length and unsupervised features extracted from the network itself. The difference is fairly small, but is also understandable, since the unsupervised features such as the number of common neighbors might have helped the random forest model to better predict the mean speed, since nodes that are close together will likely have a similar mean road speed.

To our surprise, the approach combining Node2Vec features, our second approach, with two different models produced the worst results, even worse than the baseline approach that uses only the road attributes as its features. As we mentioned in the methodology, we used different parameter combinations but neither the BFS nor the DFS approaches, with or without the edge weights, resulted in any significant difference in Node2Vec results. The MSE metric changed by at most one point for both models. Specifically, the best Node2Vec embeddings were created when the mean speeds were used as edge weights, and with $p = 1$ and $q = 10$.

We hypothesized two reasons behind the bad performance of Node2Vec: the sparseness of the graph as well as the directed nature of the graph. Both of these reasons probably caused the Node2Vec algorithm to return either wrong embeddings by traversing the directions that are not allowed in the graph; or not so helpful node embeddings, since there are too many nodes and too few edges, which resulted in too many isolated nodes. With such loosely connected nodes, the algorithm might not have been able to start meaningful random walks, thus resulting in embeddings that confused our regression models.

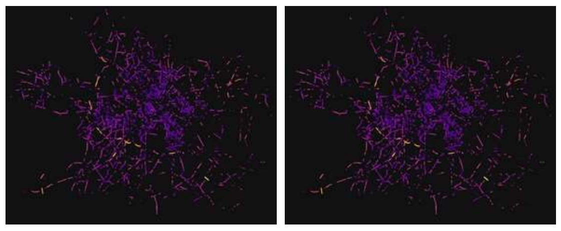
One last item worth mentioning is the superiority of our non-linear random forest model to linear regression model. This difference was the most significant for Node2Vec features, but it was also significant when we introduced features in our first approach based purely on the graph structure itself. This means that the relation between our target value, the mean speed, and the best set of features we used to predict it is quite non-linear.

Besides the test set results, we've also included a side-by-side comparison of our whole Berlin road network colored by the true mean speed of the roads, and the predicted mean speeds returned by the three random forest models we used, with each of them using a different set of features:



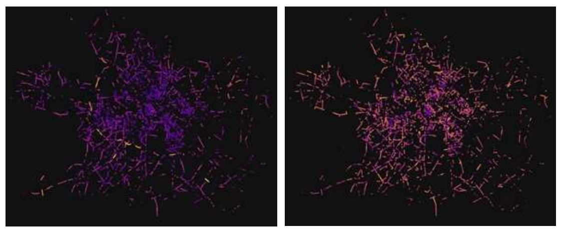
Left: Reference Graph: Roads are colored according to true mean speeds.

Right: Baseline Results: Roads are colored according to the predicted mean speeds of baseline.



Left: Reference Graph: Roads are colored according to true mean speeds.

Right: Roads are colored according to the predicted mean speeds from the first approach.



Left: Reference Graph: Roads are colored according to true mean speeds.

Right: Roads are colored according to the predicted mean speeds from the second approach.

As can be seen from the mean square error values, the color difference between RF+baseline features and RF+baseline+unsupervised graph features are so insignificant that it is hard to notice which approach performed the best just by visual inspection.

On the other hand, one can see a clear difference between the two approaches above and the RF+Node2Vec features, which we defined as our second approach.

VI. Conclusions

In this project, we aimed to create a model for predicting traffic congestion in the vehicle road network for the city of Berlin, but we failed in our goal because of the sparseness of the eventual road network we were able to create. As a result, we shifted our aim to predicting just the edge weights, namely mean speeds of the roads, in the timeframe between 7 and 10 AM from January 1 to April 1, 2020.

We were able to achieve the lowest mean square error by using the road features originally presented to us by both OSM+Uber movement datasets, plus the basic features we extracted our graph in an unsupervised manner. These features are then fed into a random forest regression model to generate predictions.

The main takeaway from our project pertains to the dataset. In order to model traffic congestion, obtaining a denser road network data, whether it belongs to the city of Berlin or another city, would be the first goal of further work. In addition, applying an embedding algorithm that can work on directed graphs, whether as a modified version of Node2Vec, can generate valuable insights since we would be able to test if using only the graph topology for feature extraction would be good enough for modeling.

VII. References

[1] M. Zhang, Y. Chen, "Link prediction based on graph neural networks." arXiv preprint arXiv:1802.09691 [cs.LG], 2018.

[2] L. Lu, T. Zhou, "Link prediction in complex networks: A survey." Physica A 390, 2011, pp. 1150-1170.

[3] B. Yu, H. Yin, Z. Zhu "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting." arXiv preprint arXiv:1709.04875 [cs.LG], 2018.

[4] T. Jepsen, C. Jensen, T. Nielsen, "Graph convolutional networks for road networks." arXiv preprint arXiv:1802.09691 [cs.LG], 2020.

[5] M. Schlichtkrull, T. Kipf, P. Bloem, R. van den Berg, I. Titov, M. Welling, "Modeling relational data with graph convolutional networks." arXiv:1703.06103 [stat.ML], 2017.

[6] X. Zhang, N. Brugnone, M. Perlmutter, M. Hirn, "MagNet: A magnetic neural network for directed graphs." arXiv:2102.11391 [cs.LG], 2021.

VIII. GitHub

https://github.com/Stackeduary/Network-Science-Project-Spring-2021