Machine Learning within a Graph Database: A Case Study on Link Prediction for Scholarly Data

Sepideh Sadat Sobhgol[1][a], Gabriel Campero Durand[2][b], Lutz Rauchhaupt[1][c] and Gunter Saake[2][d]
[1]*Ifak e.v.Magdeburg, Germany*
[2]*Otto-von-Guericke University, Magdeburg, Germany*

Keywords: Graph Analysis, Network Analysis, Link Prediction, Supervised Learning.

Abstract: In the combination of data management and ML tools, a common problem is that ML frameworks might require moving the data outside of their traditional storage (i.e. databases), for model building. In such scenarios, it could be more effective to adopt some in-database statistical functionalities (Cohen et al., 2009). Such functionalities have received attention for relational databases, but unfortunately for graph-based database systems there are insufficient studies to guide users, either by clarifying the roles of the database or the pain points that require attention. In this paper we make an early feasibility consideration of such processing for a graph domain, prototyping on a state-of-the-art graph database (Neo4j) an in-database ML-driven case study on link prediction. We identify a general series of steps and a common-sense approach for database support. We find limited differences in most steps for the processing setups, suggesting a need for further evaluation. We identify bulk feature calculation as the most time consuming task, at both the model building and inference stages, and hence we define it as a focus area for improving how graph databases support ML workloads.

1 INTRODUCTION

Authors commonly define *Machine learning* (ML) as the process of solving a problem by relying on a dataset and building a statistical model over it (Burkov, 2019). With the continuous digitization of everyday life, ML has become an increasingly popular way for companies to extract business insights from their copious data, and to offer new services (Shoham et al., 2018). Instead of having to move a large amount of data outside of databases, into another tool, for building ML models, it would be much more convenient and efficient to have a single platform in which the traditional storage and the novel processing can be both embedded. Hence, integrating ML algorithms into existing database systems so that we can take advantage of the benefits in both technologies is an active area of research, specially in relational databases (Kumar et al., 2017). However the development of such extensions for databases with non-relational models (e.g. document or graph models) has not received similar interest at this stage. Graph databases are a type of database system that allows to store entities as nodes and edges in a graph data model. In addition these systems offer specialized query languages and plugin libraries to express queries from the field of network science (e.g. connected components, shortest paths, etc.). Considering analysis, graph databases provide natural support for network-based feature engineering, potentially helping ongoing ML tasks in finding more context for data items (by considering their relationships). These observations on: a) the potential of graphs to improve ML processes, and b) existing limitations in tooling to support such processes, serve as compelling motivations for research.

In this work we aim to identify how a general case study might look like. We also seek to evaluate from a practical perspective, the pipeline of running classifiers on graph databases, compared to running them outside of the database system, and to identify in an early stage, the stress points of the process. From our research we are interested in execution time, F-scores of the trained models (to assess the feasibility of the model), and complexity of implementation.

The ML task that we choose for our study is link prediction. Link prediction is defined as the

[a] https://orcid.org/0000-0002-9746-3612
[b] https://orcid.org/0000-0002-4901-1849
[c] https://orcid.org/0000-0002-6907-0520
[d] https://orcid.org/0000-0001-9576-8474

Sobhgol, S., Durand, G., Rauchhaupt, L. and Saake, G.
Machine Learning within a Graph Database: A Case Study on Link Prediction for Scholarly Data.
DOI: 10.5220/0010381901590166
In *Proceedings of the 23rd International Conference on Enterprise Information Systems (ICEIS 2021) - Volume 1*, pages 159-166
ISBN: 978-989-758-509-8

task of predicting the (future) possible connections among nodes in different networks (Agarwal et al., 2012), given features from the current state of the network. This task is a very common ML task on graphs, and it has applications in social networks, chemistry and other domains.

To support such a task in- and outside a graph database, we require a process that follows the next steps:

1. **Data Preparation**, which includes train and test data split.
2. **Model Building**, covering training and storing a model inside the database.
3. **Model Exploration**, which includes the exploration of the model by utilizing meta-data. This is beyond the scope of this research.
4. **Inference**, which includes using the model to make new predictions, given data passed as an example.

After defining the steps of the process, we can now establish the main contributions of this study: a) We provide a simple prototype for link prediction using traditional network features and XGBoost, over a subset of the Microsoft Academic Graph. [1]. b) We compare the costs of data split and feature calculation using an off-the-shelf database, and in- and out-database processing. c) We validate the feasibility of our pipeline, with limited hyper-parameter tuning and feature importance analysis. d) We report early results for inference in- and outside the database. The remainder of this paper is structured as follows: Sec. 2 provides basic concepts for graph ML tasks and link prediction. Sec. 3 introduces our research questions and the setup for our study. We follow with our results in Sec. 4 and we wrap-up with our conclusions and suggestions for future research, in Sec. 5.

2 BACKGROUND

In this section, we briefly establish key terms and context to understand our study. We start with a basic listing of ML tasks on graph data and ML-driven approaches towards link prediction.

2.1 Machine Learning Tasks On Graphs

The most common ML tasks on graphs that can be found in the literature include Link Prediction, Community Detection (Agarwal et al., 2012), Node Classification (Agarwal et al., 2012), Anomaly Detection (Kaur and Singh, 2016; Chu et al., 2012) and Graph Embedding (Wang et al., 2016; Goyal and Ferrara, 2018). In traditional research graph features are calculated and then fed to ML models such as SVMs, or Random Forests (Agarwal et al., 2012), in recent years the use of graph neural networks has been popularized, leading to improvements over more traditional models (Dwivedi et al., 2020).

The task of link prediction is to predict the possibility of a future connection between nodes by using features of the entities in the current graph (Agarwal et al., 2012; Vartak, 2008). Current link prediction algorithms can be categorized into supervised (Tang et al., 2016), unsupervised (Kashima et al., 2009) and semi-supervised (Berton et al., 2015) methods.

2.2 Machine Learning Approaches towards Link Prediction

For a set of training data, if there is a link between two vertices, the data point will get a label +1 otherwise -1. A classification model which is based on this definition needs to predict a label for data points which are not currently linked in the train dataset. Classification models can be built based on a set of features, such as so-called graph-topological features, including Node Neighborhood-Based Features (based on this feature, the probability of having connection between two nodes x and y grows if the number of neighbours they share increases), Path-based Features and Features based on Node and Edge Attributes (Agarwal et al., 2012; Mutlu and Oghaz, 2019).
Different supervised learning algorithms have been used in the literature for this classification including naive Bayes, k-nearest neighbors, neural networks, support vector machines and XGBoost. Comparing the performance of different classification algorithms on link prediction task, shows that XGBoost (Becker et al., 2013) is an appropriate solution especially when we deal with unbalanced data (Chuan et al., 2018). Based on (Al Hasan et al., 2006), the performance of the mentioned algorithms on BIOBASE and DBLP datasets are shown to not be as good as the performance of XGBoost. Apart from traditional supervised algorithms, there are also other emerging approaches towards link prediction. For example a state of the art solution for finding relations between semantic concepts on the WN18RR dataset has been presented by Pinter and Einstein (Pinter and Eisenstein, 2018). Other recent models for link prediction include DihEdral (Xu and Li, 2019), InteractE (Vashishth et al., 2019), HypER (Balazevic et al., 2018), ConvE(Dettmers et al., 2018) and De-

[1]The code is available online https://github.com/sepideh06/ML-on-graphs/

Com (Kong et al., 2019) which are all designed based on neural networks. Such approaches rely more on neighborhood information and less on traditional graph topological features.

3 LINK PREDICTION ON A SUBSET OF THE MICROSOFT ACADEMIC GRAPH

In this section, we introduce our research questions. We also describe the key components of our approach including our dataset, and the studied configurations for the ML process with a graph database.

3.1 Research Questions

To guide our research to support a standard ML process over a graph model, we have chosen a list of research questions:

1. **Data Preparation.** What is the impact, in terms of execution time, of performing data splitting and feature calculation in- and outside of the database?
2. **Model Building.** How efficient is it to build, store and evaluate a model in- and outside the Neo4j graph database? How relevant are different combinations of graph features to improve the accuracy (efficacy) of the selected link prediction model?
3. **Inference.** How does the size of data which is used for inference affect the performance of the task?

3.2 Dataset

The dataset we have selected to use, is a subset of the Microsoft Academic Graph (Sinha et al., 2015), (published on 2018-11-09). This dataset contains information about scientific publications, journals, conferences, the relationships between publications, authors, affiliations and the relationships between authors and papers.

Fig. 1 shows a Co-authorship network which is a type of bibliographic network, such as the MAG. Nodes represent the relations between papers, venues, topics and authors (Lee and Adorna, 2012).

We choose publications and their relationships with authors for our link prediction task to predict possible future collaboration between authors. Since as we get close to 2019, the number of publications gets quite large, for our study we decided to choose

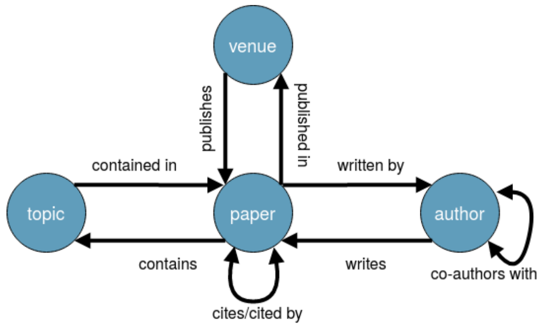

Figure 1: Schema of co-authorship network (based on (Lee and Adorna, 2012)).

all publications as of selected years (we report using the year 1989). For the first year considered we label publications as train and test both, with the links of the following year considered for the train labels. For the test data we have used the data of both the first and second year, with the links of the third year used as test labels. We did not consider information from years previous to the one taken as the first year. Our dataset consists of 6.7 M papers, distributed as follows for the three years (1989, 1990, 1991): 2.1 M, 2.3 M, 2.3 M, and 10 M authors. In terms of relationships, the dataset contains 16.5 M authorship relationships, distributed as follows for the selected years: 5.2 M, 5.6 M, 5.7 M. It also contains 54 M co-authorship relationships, distributed as follows: 15 M, 20 M, 18.7 M, and 3 M citation relationships among the papers of the corresponding years. We report the information for co-authorships in duplicate, to account for the un-directed nature of edges. Otherwise there are, 27 M edges for co-authorships in the selected years.

Fig. 2 shows a visualization of the co-authorship graph for the years we study (based on a random sample of 50k edges), using the OpenOrd (Martin et al., 2011) algorithm for graph layouting as provided in Gephi. The figure shows nodes (authors) with colors and sizes proportional to the number of edges (co-authorships) connected to them. The figure shows a general pattern where most authors have a small number of co-authorships on the year, creating the multiplicity of blue leaf-like relationships that make-up the largest area of the graph. At the center, with darker colors, there are small numbers of authors forming very dense communities of collaborations.

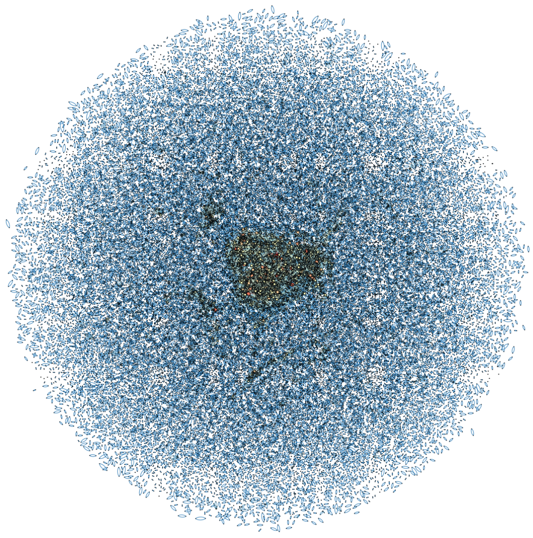

Figure 2: Visualization of Co-authorship Graph for 1991, based on Gephi and OpenOrd Layouting algorithm.

3.3 Link Prediction with a Graph Database: A Stored Procedure-based Approach

In this section we introduce the different execution scenarios we consider for involving a graph database in the ML task. For this paper we choose Neo4j as our graph database. It is a popular database supporting property graphs, equipped with the Cypher query language. Executing the process from outside of a database is straight-forward: the database can be used for storing the intermediate data between steps (e.g. test and train splits, and the model itself), and the interaction with the database can be accomplished through a program developed by a user that employs a client from the database with its API for database interaction. The execution from inside the database follows a similar logic, but the code with the client connection has to be shipped beforehand to the database, as a stored procedure. This stored procedure can then be called from outside, as a request to the database, and its execution starts and concludes on the server side. Fig. 3 describes the generic process of doing link prediction in- and outside the database. The feature calculation (offline) stands for those features which can be independently calculated for each node such as PageRank (which is used to compute the rank of a node by iteratively propagating node importance over the graph). There are also some other features such as 'common neighbours' or 'shortest path' which are required to be calculated on the fly because they arefor a specific pair of nodes (for example a pair of authorIDs) and need to be computed online.

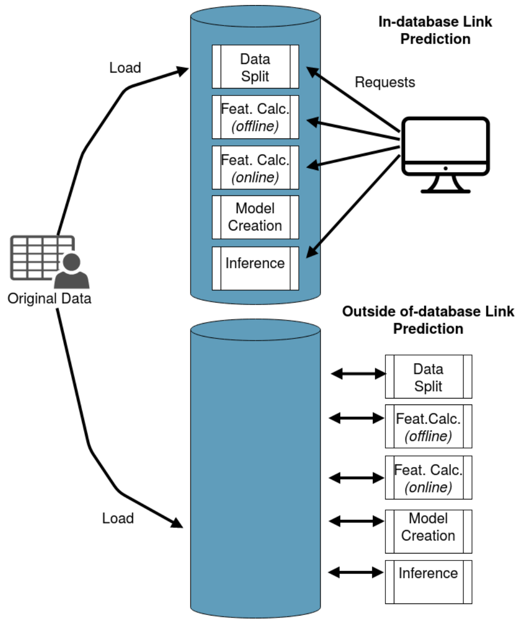

Figure 3: In-Database and Outside of Database Link Prediction.

4 EVALUATION

4.1 RQ1: Data Preparation

In this section, we provide experimental insights to answer to the first of our research questions.

4.1.1 Creating Balanced Training and Testing Datasets

With link prediction, we are interested in predicting the possible future collaboration between authors. This dataset provides a good criteria for splitting train and test data, by using the 'Year' for each publication. However existing links need to be complemented with a selection of non-existing links (for training on both kinds of labels). To this end we randomly sample without repetition from the set of all possible non-existing links, until we match the number of positive examples in the corresponding split. We make two different lists of train and test pairs (just the ids of the nodes), and we save the lists back in the database, as properties of two nodes named train and test. These nodes are further tagged, with a timestamp and additional metadata, to facilitate keeping track of ML experiments. In order to split data efficiently, a stored procedure is required to be inserted into Neo4j as a plugin.

4.1.2 Feature Calculation

For our dataset, we consider five features including 'common neighbours', 'shortest path in co-author graph', 'second shortest path in co-author graph', 'shortest path in citation graph' and 'second shortest path in citation graph'. We calculate these features because previous work establishes that some similarity measurements (Al Hasan et al., 2006; Han and Xu, 2016), such as shortest path and common neighbours., have been shown to contribute to good model performance. For example the shortest distance between two authors will be calculated by counting the number of hops between them. This captures the idea that perhaps a smaller distance between authors could be an indicator of their possibility of collaborating in the future. All the features can be calculated by using Cypher queries.

In order to understand the density of our data, we calculate the distribution of our features for positive and negative observations. We employ cumulative distribution plots to describe a summary of the connection between authors in terms of different features. Fig. 4 and Fig. 5, the distribution plots for train and test data for the feature 'Shortest Path in co-Author graph' are presented. For train data in Fig. 4 and for 'exist' connections, -1 is representative of no shortest path. The Cypher query return -1 instead of 0, if there is no shortest path between a pair of author. The plot depicts that a value of 2 hops covers more than 300 authors and very small number have a shortest path greater than 20. For 'non-exists' connections, there is no shortest path between more than 400 authors. A very small portion have a shortest path greater than 20. The distribution plots for train and test data for the other features repeats similar patterns concerning a clear difference in the distributions between labels (signalling the common-sense nature of the selected features), and stability in label distributions across the years.

4.1.3 Experiment Result

This process of reading pairs of authors and calculating five features for each pair is quite time consuming, so for the sake of repeated experiments, we sampled 2k random observations at a time. Table. 1 shows the recorded time for splitting data and calculating features in- and outside the data base. Data split inside the database takes 2 times less than doing the process outside the database. In addition, there are smaller relative performance improvements in feature calculation for in-database usage.

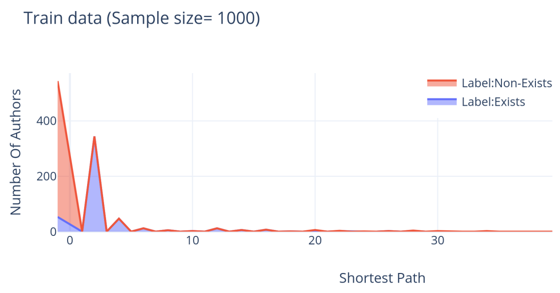

Figure 4: Shortest Path-CoAuthor Graph-Train Data.

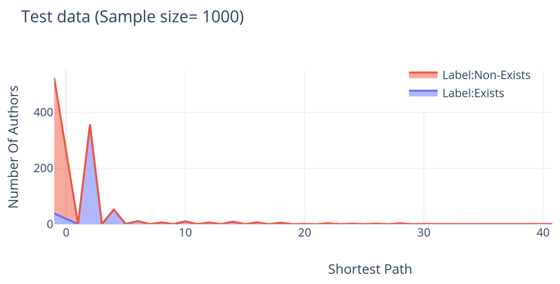

Figure 5: Shortest Path-CoAuthor Graph-Test Data.

Table 1: Data Split and Feature Calculation In- and Outside Neo4j.

	In-	Outside Graph DB
Data Split	3.1 minutes	7.2 minutes
Feature Calculation	15.21 hours	16.93 hours

4.2 RQ2: Model Creation

In this section we present the results of our experimental evaluation on the aspects of model building, addressing the second of our research questions. In order to create and train a model, a list of parameters needs to be set which will be later passed to the train function of XGBoost. The key parameters which affect the most the performance of the model are 'learning_rate', 'n_estimators' and 'early_stopping_rounds'. The last parameter will make the algorithm stop if after a certain number of iterations, no improvement in 'log loss' or 'classification error' of algorithm is obtained. The goal is to minimize 'mlog loss' by setting other parameters such as 'n_estimators' and 'learning_rate' to some optimal values.

Fig. 7 is the result of including all 5 features and having 'learning-rate' equal to 0.01. The minimum log loss is 0.483624 which is obtained at iteration 99. By changing 'learning-rate' to 0.001 and 'n-estimators' to 500,700,1000, we get higher mlog loss equal to 0.490644. The second attempt in which the 'learning-rate' is equal to 0.01 generates the minimum error for our dataset. The value we consider for

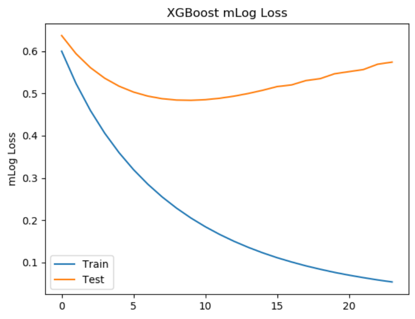

Figure 6: XGBoost mlog loss for 24 iterations-0.1 learning rate-5 features.

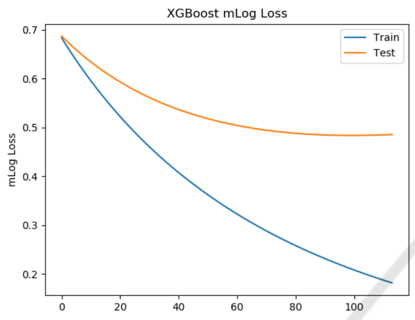

Figure 7: XGBoost mlog loss for 100 iteration-0.01 learning rate-5 features.

mlog loss is based on test set (orange line), since the result for train set is so optimal which signals that we could be overfitting. In the next step, we would like to know which features are more valuable in building the decision trees within the XGBoost model. The importance score can be calculated for each feature, helping the features to be ranked and prioritised within the dataset. For calculating importance of each feature, we can make use of the Gini Index. Feature importance give us an array of values for our 5 features. Table.2 shows the values for each feature in Threshold column. Common neighbour and second shortest path in citation graph have the least values and hence can be considered as the most important features. 'SelectFromModel' class which belongs to 'scikit-learn'library in Python will consider each score as a threshold. Applying threshold will help to filter one or more features in each step to observe if the performance of the model improves. Considering all features, first of all we train a model using our train dataset, then measure the performance of the model on the test data considering the same feature selection. Table.2 shows that the performance of the model does not change by decreasing the number of selected features until the last step , in which F-score decreases from 81.65% to 76.98%. Apart from reducing the features based on threshold, we reduce the number of features by removing the features randomly to see how it affects the accuracy of the model. The performance of the model increases by decreasing the number of selected features to 'common neighbours' and 'shortest path in co-author graph' from 81.65% in the first step of having 5 features to 94.43%. Hence our results show us that, contrary to expectations, a smaller set of features is sufficient for improving the performance of the model. Table.3 shows the result. Fig. 8 is the result of including 2 features (common neighbors and shortest path in co-authorship graph) and having 'learning-rate' equal to 0.01 and 'n-estimate' equal to 500. The minimum log loss is obtained which is equal to 0.1789333.

Table 2: Filtering Features based on Threshold For 5 Features.

Threshold	Num Of Features	Accuracy	F-Score
0.000	5	77.70%	81.65%
0.004	4	77.70%	81.65%
0.005	3	77.70%	81.65%
0.035	2	77.70%	81.65%
0.955	1	70.40%	76.98%

Table 3: Filtering Features based on Threshold For 2 Features.

Threshold	Num Of Features	Accuracy	F-Score
0.090	2	94.60%	94.43%
0.910	1	94.60%	94.43%

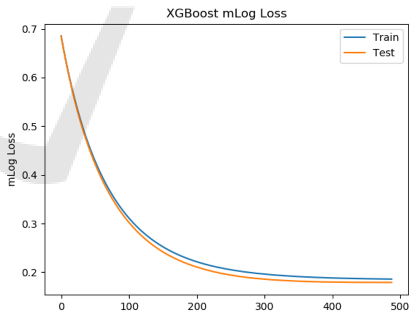

Figure 8: XGBoost mlog loss for 500 iteration-0.01 learning rate-2 features.

We include this analysis, just to illustrate the role of parameters and features in optimizing our results. After calculating features for each pair of authors, we need to train the model based on XGBoost and store it inside Neo4j. The detailed implementation can be found in GitHub[2]. The results show that training

[2]See the storedprocedure named 'CreateAuthorPaperModel' inside the class 'AuthorPaperModel' in: https://github.com/sepideh06/ML-on-graphs

Table 4: Inference In- and Outside Neo4j.

	In-	Outside the Graph
Feature Calculation	3.31 hours	3.16 hours
MI For 100 Samples	111 ms	104 ms
MI For 75 Samples	94 ms	113 ms
MI For 50 Samples	92 ms	95 ms
MI For 25 Samples	97 ms	99 ms

the model on the database side, using a stored procedure, can be close-enough in terms of execution time, as training the model on the user side and the difference between the approaches is not markedly large, though more studies are required. The evaluation results also reveal that features such as shortest path are quite informative for link prediction task and result in better performance measurements.

4.3 RQ3: Inference

In this section we present the results of our experimental evaluation on the aspects of inference, using our experimental prototype. Inference involves using the learned model to make a prediction, given some data passed as an example. If the model is saved within Neo4j, we need to retrieve it and read it into an array of bytes, interpreting it then into the API of the ML library on which the model is implemented. From outside of the database, the model is already saved as a local file, hence, we just need to load the model, ask the model about our sample and record the time for the predictions. Table. 4 presents an overview on recorded times for different samples over 10 iterations. Evaluation results show that increasing the size of the sample does not deteriorate the performance of the model in- or outside of graph database Neo4j. Although, the recorded time for the batch will be increasing little by little as the size of data is growing but it does not create such a big difference in the final performance. Besides, there is not huge difference in recorded time of doing inference in the alternative cases, since for both situations, such time is spent to retrieve the model from either inside database or a local file. The feature calculation for the new data, is shown to be at an entirely different temporal scale than inference.

5 CONCLUSION

In this paper, we provided an early consideration of what could be a practical case study and design choices for carrying out an ML task with the support of an off-the-shelf graph database. We envision that the graph database could be tasked to help with feature calculation, and to store metadata about the model which assist model exploration and inference. To evaluate some choices for such a support we studied link prediction with traditional ML models (XGBoost) and simple graph-based features, over a subset of the popular Microsoft Academic Knowledge Graph, using the Neo4j graph database with a stored-procedure-based integration. From our study we find that feature calculation is by far the most time-consuming task of the process, with an impact on inference and model building. Hence for graph databases and ML based on traditional graph features, future research should emphasize combinations of the database with scale-out processing frameworks (e.g. Neo4j and Spark), and extensions to query interfaces to support better bulk feature calculation. The latter improvement shares research interests with ongoing work in close couplings of ML tasks with relational databases (Ngo and Nguyen, 2020). Regarding this concern we should also note that specific algorithmic improvements building on fine-grained measurements, scoped per type of feature and kind of topology, would contribute to the research. Based on our work we can also suggest that for traditional ML, a sample-based approach for evaluating feature combinations prior to training models on the whole data is a common-sense important practice to focus the processing on the highest-leverage features. The automation of the aforementioned practice could also be of interest for process improvements. In our study we identified that only 2 features could produce the best F-score, illustrating the need to establish feature relevance so as to limit the computational requirements of feature calculation. In our work we also provide an early assessment on the time for model training when the model is in- or outside of the database. We find little difference between both cases, when not considering feature calculation as part of the process. Regarding inference, we find that there is a small advantage of the in-database option over the alternative, similar observations were made for data splits, but we identify that future studies with finer-grain profiling and more data size variations are required to better understand these findings. Moving forward, we suggest that as graph neural networks are becoming increasingly competitive for ML tasks, cursor-based approaches for dataset/feature generation in order to produce the tensors required to train/use such models, constitutes a potentially rewarding novel workload for graph data management systems to consider.

REFERENCES

Agarwal, A., Corvalan, A., Jensen, J., and Rambow, O. (2012). Social network analysis of alice in wonderland. In *Proceedings of the NAACL-HLT 2012 Workshop on computational linguistics for literature*, pages 88–96.

Al Hasan, M., Chaoji, V., Salem, S., and Zaki, M. (2006). Link prediction using supervised learning. In *SDM06: workshop on link analysis, counter-terrorism and security*.

Balazevic, I., Allen, C., and Hospedales, T. M. (2018). Hypernetwork knowledge graph embeddings. *arXiv preprint arXiv:1808.07018*.

Becker, C., Rigamonti, R., Lepetit, V., and Fua, P. (2013). Supervised feature learning for curvilinear structure segmentation. In *International conference on medical image computing and computer-assisted intervention*, pages 526–533. Springer.

Berton, L., Valverde-Rebaza, J., and de Andrade Lopes, A. (2015). Link prediction in graph construction for supervised and semi-supervised learning. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Burkov, A. (2019). *The Hundred-Page Machine Learning Book*. Andriy Burkov.

Chu, Z., Widjaja, I., and Wang, H. (2012). Detecting social spam campaigns on twitter. In *International Conference on Applied Cryptography and Network Security*, pages 455–472. Springer.

Chuan, P. M., Giap, C. N., Bhatt, C., Khang, T. D., et al. (2018). Enhance link prediction in online social networks using similarity metrics, sampling, and classification. In *Information Systems Design and Intelligent Applications*, pages 823–833. Springer.

Cohen, J., Dolan, B., Dunlap, M., Hellerstein, J. M., and Welton, C. (2009). Mad skills: new analysis practices for big data. *Proceedings of the VLDB Endowment*, 2(2):1481–1492.

Dettmers, T., Minervini, P., Stenetorp, P., and Riedel, S. (2018). Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. (2020). Benchmarking graph neural networks. *arXiv e-prints*, pages arXiv–2003.

Goyal, P. and Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94.

Han, S. and Xu, Y. (2016). Link prediction in microblog network using supervised learning with multiple features. *JCP*, 11(1):72–82.

Kashima, H., Kato, T., Yamanishi, Y., Sugiyama, M., and Tsuda, K. (2009). Link propagation: A fast semi-supervised learning algorithm for link prediction. In *Proceedings of the 2009 SIAM international conference on data mining*, pages 1100–1111. SIAM.

Kaur, R. and Singh, S. (2016). A survey of data mining and social network analysis based anomaly detection techniques. *Egyptian informatics journal*, 17(2):199–216.

Kong, X., Chen, X., and Hovy, E. (2019). Decompressing knowledge graph representations for link prediction. *arXiv preprint arXiv:1911.04053*.

Kumar, A., Boehm, M., and Yang, J. (2017). Data management in machine learning: Challenges, techniques, and systems. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1717–1722. ACM.

Lee, J. B. and Adorna, H. (2012). Link prediction in a modified heterogeneous bibliographic network. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, pages 442–449. IEEE Computer Society.

Martin, S., Brown, W. M., Klavans, R., and Boyack, K. W. (2011). Openord: an open-source toolbox for large graph layout. In *Visualization and Data Analysis 2011*, volume 7868, page 786806. International Society for Optics and Photonics.

Mutlu, E. C. and Oghaz, T. A. (2019). Review on graph feature learning and feature extraction techniques for link prediction. *arXiv preprint arXiv:1901.03425*.

Ngo, H. Q. and Nguyen, X. (2020). Learning models over relational data: A brief tutorial. In *Scalable Uncertainty Management: 13th International Conference, SUM 2019, Compiègne, France, December 16–18, 2019, Proceedings*, volume 11940, page 423. Springer Nature.

Pinter, Y. and Eisenstein, J. (2018). Predicting semantic relations using global graph properties. *arXiv preprint arXiv:1808.08644*.

Shoham, Y., Perrault, R., Brynjolfsson, E., Clark, J., Manyika, J., Niebles, J. C., Lyons, T., Etchemendy, J., and Bauer, Z. (2018). The ai index 2018 annual report. *AI Index Steering Committee, Human-Centered AI Initiative, Stanford University. Available at http://cdn. aiindex. org/2018/AI% 20Index*, 202018.

Sinha, A., Shen, Z., Song, Y., Ma, H., Eide, D., Hsu, B.-j. P., and Wang, K. (2015). An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*, pages 243–246. ACM.

Tang, J., Chang, Y., Aggarwal, C., and Liu, H. (2016). A survey of signed network mining in social media. *ACM Computing Surveys (CSUR)*, 49(3):42.

Vartak, S. (2008). A survey on link prediction. *State University of New York*.

Vashishth, S., Sanyal, S., Nitin, V., Agrawal, N., and Talukdar, P. (2019). Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions. *arXiv preprint arXiv:1911.00219*.

Wang, D., Cui, P., and Zhu, W. (2016). Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM.

Xu, C. and Li, R. (2019). Relation embedding with dihedral group in knowledge graph. *arXiv preprint arXiv:1906.00687*.