Large Scale Supervised Link prediction with GraphX and Spark MLlib

Mohammad Khamar Uz Zama
Data and knowledge Engineering
Otto von Guericke University
Magdeburg,Germany

Suraj Shashidhar
Digital Engineering
Otto von Guericke University
Magdeburg,Germany

Pramod Kumar Bontha
Data and knowledge Engineering
Otto von Guericke University
Magdeburg,Germany

Madhu Gopal Sirivella
Digital Engineering
Otto von Guericke University
Magdeburg,Germany

Avinash Ranjan
Digital Engineering
Otto von Guericke University
Magdeburg,Germany

Sanjeeth Busnur Indushekar
Digital Engineering
Otto von Guericke University
Magdeburg,Germany

***Abstract*—Graph data has been gaining a lot of attraction in recent years. However, efficient support for machine learning, in unison with its data management, is under-researched for the graph domain. Using this as our motivation we consider in this paper a representative machine learning task on graph data, while using state-of-the-art graph tools. We work on link prediction over a Co-Authorship graph, from scientific publications. In our work we identify a set of graph features to support the learning task. As part of feature evaluation, we perform a cost-based analysis of the features extraction across Spark and Neo4J frameworks. After calculating the features, we study the performance of a set of classifiers for the task, comparing them using F1 scores. As part of our conclusions, we establish which combination of features and classifiers gives best results, helping researchers to reason about the trade-off between feature computation and feature contribution to the task. From our observations, Pagerank with Connected components combined were shown to be the best features with an average F1-score *0.75954* across all the classifiers and the Gradient Boosted tree classifier performed better with an average F1-score *0.64710* across all feature combinations.**

***Index Terms*—Graph Data, Large scale Graph Machine Learning, Spark, Graph Features, Neo4J, Framework evaluation, Feature evaluation.**

I. Introduction

Graphs, a mathematical model that consists of a set of vertices and edges connecting them, can be applied everywhere. From representing a network of interconnected IoT devices, to fraud detection, to recommendations in retail or social media, there seems to be an application in every domain which can be modelled in the form of a graph. Graphs have been gaining attention as many real-world systems can be easily represented as connected networks where each vertex represents an object and the edge describing the nature of the relationship of that vertex with other vertices. But apart from the ease of representation, interesting properties of the real-world can be understood from the graph representation, and hence graphs are a mechanism to extract more information from real-world data. Machine learning, the area of research in charge of creating statistical models to describe the behavior of datasets, is important to help businesses gain insights into large data. It can also be performed on graphs to discover hidden and new patterns, identifying communities, classifying nodes and link prediction using graph-structured data as feature information.

There are many machine learning applications on graphs, including node classification, community detection, among others. Link prediction is one of the key areas under research on graphs. Link prediction aims to predict if two nodes in the graph will form an edge in future. Hence link prediction can be simply expressed as a form of binary classification where the goal is to classify if two nodes will connect in future or not. This can be achieved by analyzing and representing the current and past behaviour of pairs of nodes in the form of a feature vector and feeding this vector to a traditional machine learning algorithm. Traditional and ensemble classifiers can be used.

Performing Machine Learning usually requires a huge amount of data to give acceptable results, since this reduces the chance of using a non-representative sample. This is also true in graphs. Hence, machine learning on graphs can also be data management problem. Although there exist large scale graph

processing frameworks, and graph databases, to our knowledge they have not been studied thus far in terms of their support for machine learning pipelines. Hence, in this work we research on a representative graph task (link prediction), on a large public dataset (the *Microsoft Academic Knowledge Graph* [8], hereafter referenced as MAKG), while using a generic set of features, a large scale graph processing engine (GraphX from Spark, but we also compare with a single-node graph database configuration, Neo4j) and a simple set of classifiers. Through this approach we are able to analyze the contribution of features to the learning task, their execution times and some scalability limits of the tools we use.

To summarize, our contributions in this research work are:

1) We evaluated different models and combinations of graph features, on how well they perform link prediction using F1 scores.
2) We evaluated how the feature calculation step of link prediction with a large scale graph processing framework, using GraphX, compares in performance with respect to a graph database, Neo4J.
3) We categorize features based on their calculation cost and contribution towards achieving the goal.

The rest of this paper is organized as follows:

Section 2 gives detailed information on graphs and features required for understanding the Link Prediction. Section 3 describes the design and architecture of our prototype for the machine learning pipeline implemented in this work for link prediction. It also gives information on dataset schema, and how we have partitioned the data, into test and train. Section 4 discusses the experimental setup, extraction of features, more on dataset and evaluation measures. Section 5 presents the results of the evaluation of each of the classifiers for different combinations of features and runtimes. In section 6 we discuss a small number of papers of related research on machine learning on graphs. Finally, Section 7 concludes and discusses future work.

II. Background

Graph structures usually consider only one type of node and edge i.e. the single relationship graph. For practical applications, we should be able to distinguish between different kind of node and edges, this means that we need a well-defined graph data model. A property graph is such model. It is defined as labelled and attributed directed multi-graph with identifiers. The data is organised clearly as nodes, edges (relation between the nodes) and properties. The node represents the entities that hold the number of attributes called properties. An edge represents the relationship which provides directed, named, and semantic clarity between the two entities. Each Node and Edges will be having unique IDs and the attributes which define the properties of nodes and edges.

Different types of graphs:

1) Simple graph - Graph where two vertices are joined by a single edge without loop formation, or without duplicate edges between two nodes.
2) Edge-labeled graph - Graph which represents the relationship between the two vertices. Here the labels define the kind of relationship.
3) Undirected graph - Graphs where edges are undirected which is typically used when the relationship between the vertices is symmetric.
4) Directed graph - Graphs which represents the edge orientation between the vertices.

Machine learning on graphs plays a very important role in many applications. The primary challenge over here is to find a way to encode the graph structures that can be utilised by the machine learning models to provide useful results. Machine learning could be implemented in different ways on graphs such as Node classification, community detection, graph embedding and link prediction.

Link prediction is used to predict the likelihood of the existence of an unknown link or future link based on the available information knowing that there is no association between the nodes in the current snapshot of the network(graph). It is used to find hidden relationships between entities, and for predicting how the network structure will evolve in the future. Social network are one domain where link prediction is commonly used, for example to suggest new friends for the users.

To perform link prediction between two nodes, their behaviour is calculated as features which are later fed into the algorithm. These features can be broadly categorized into two:

1) **Topological features -** Features are calculated based on the graph properties such as shortest path and common neighbours.
2) **Non-topological features -** Features that are not inherent to the graph or not calculated using graph properties. Some example of such feature could be the last names of authors.

The used features of a graph such as global, local features can be used to classify Link prediction algorithms. Node structure and neighbourhood is taken care of by local features, whereas the global features use the overall path structure of the network.

A. *Global features:*

Pagerank- Pagerank is one of the best known of the centrality algorithms. It determines for each node a score that considers the number, and quality, of links to the node. This score seeks to be an estimation of how important the node is, with regards to the complete graph. The pagerank score of nodes is calculated by iteratively distributing one node's rank over its neighbours. The underlying assumption is that nodes of importance are more likely to receive a higher volume of links from other nodes. [7]

$$PR(u) = (1-d)+d(\frac{PR(T1)}{C(T1)}+...+\frac{PR(Tn)}{C(Tn)} \quad (1)$$

Where node 'u' has edges from nodes T1 to Tn, d is the damping factor which ranges from 0 to 1. For this experiment, the damping factor is set to 0.15 (1-d) is the probability that a node is reached directly without following any other edges. C(Tn) is defined as the out-degree of node T.

B. *Local features:*

Common Neighbor- Common neighbours capture the idea that two strangers who have a friend in common are more likely to be introduced than those who don't have any friends in common. The assumption is that more the number of common neighbours between two authors, more likely they will collaborate in future. The common neighbours score is calculated as:

$$CN(x,y) = \left|\Gamma(x) \bigcap \Gamma(y)\right| \quad (2)$$

where Γ(x), Γ(y) represents the set of neighbours of node x, and node y respectively, and the above expression gives the number of nodes connected to both x and y [1].

Preferential Attachment- The Preferential Attachment is a measure used to compute the closeness of nodes, based on their shared neighbours. This metric is the result of the study of Barabasi and Albert in which new nodes joining the network are proved to have a higher probability of connecting to existing nodes with high degrees rather than to nodes with lower degrees [10]. Simply stated, at any moment, people with many friends or citations, are more likely to obtain further friends or citations, than those with a smaller number of friends or citations.

$$PA(x,y) = |\Gamma(x)| \cdot |\Gamma(y)| \quad (3)$$

Taking this as a base idea we have calculated the preferential attachment score for each author by normalizing the degree of each author with factor (2x (number of edges in sub-graph)). The number 2 is taken into consideration due to bidirectional nature of our graph.

$$PA(x) = \frac{|\Gamma(x)|}{2 \times no.of edges} \quad (4)$$

C. *Path based features:*

Shortest Path-The Shortest Path algorithm calculates the shortest (unweighted) path between a pair of nodes. In this category, Dijkstra's algorithm is the most well-known. It is a real-time graph algorithm and can be used as part of the normal user flow in a web or mobile application.

D. *Community based feature:*

Connected Component- To calculate this feature, a graph is divided into discrete components such that each component forms a weakly connected sub-graph on its own, and there are no links between the not-connected components. The reasoning behind considering the connected component membership as a feature for link prediction is that authors belonging to the same community are more likely to collaborate.

E. *Graph systems:*

Graph systems include storage, processing or both. Currently soft-wares Neo4j, Tigergraph etc provides the ability to both stores the graph in its native format and process them. Graph processing frameworks such as GraphX from Spark, Gradoop, or others allow us to process graphs on large scale but not necessarily store the graph data in its native format.

III. Design

A. *The research questions we tried to answer in this paper are:*

1) How do graph databases compare with graph processing frameworks?
2) How do various features scale with respect to the size of the graph?
3) How time-consuming is each feature to calculate?
4) Which combination of features contributes better for link prediction?
5) Which classifier performs better?

B. *Workflow:*

Fig 1 gives a basic idea on the workflow of this work. The RDF dump files (obtained for a version of the MAKG made available by a researcher from the Karlsruhe Institute of Technology[1]) are first pre-processed to give usable CSV files. These files are used for generating graphs with graph libraries. From the graphs we calculate features using distributed general-purpose cluster-computing framework and a graph database. However, the computing framework is further used to partition the data into train and test sets. Finally, these datasets are now used for performing Link Prediction, with a large-scale Machine Learning processing framework (e.g., for our implementation we use Spark ML).

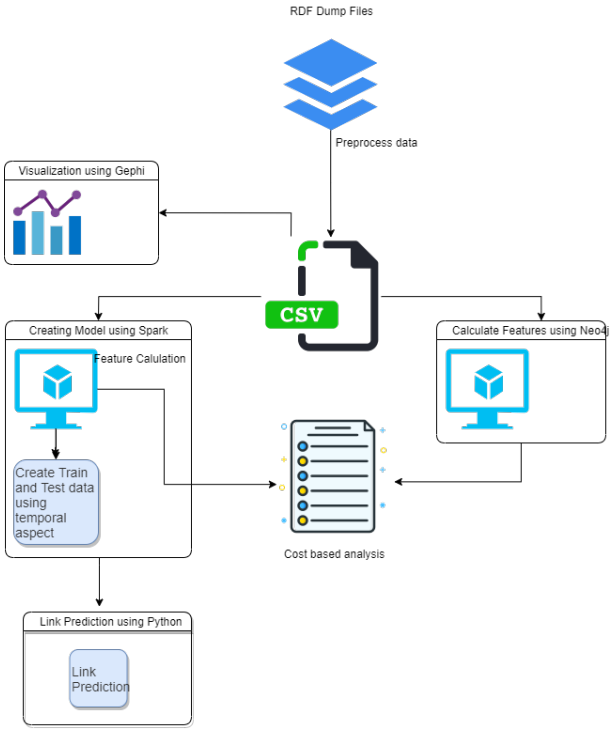

Fig. 1: Workflow of the experiment

C. *Data set*

The RDF data Microsoft Academic Knowledge Graph (MAKG) [8] contains information about scientific publications, their authors along with their respective institutions and field of study. Node types: publications (papers), keywords, authors, organizations, conferences and journals. Explicit edge types: citations, authorship (writes), organization membership, etc. Implicit edge types: co-authorships, co-citations, etc.

[1]http://ma-graph.org/

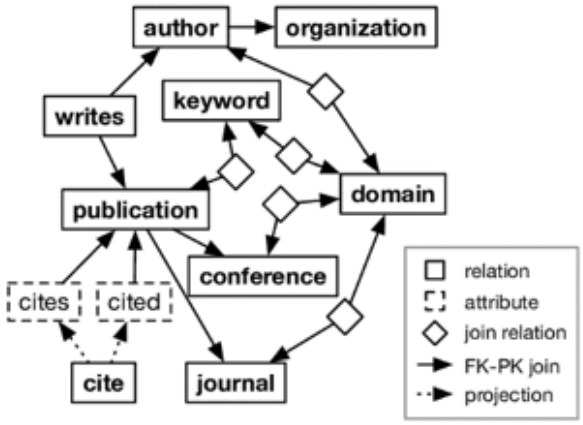

Fig. 2: Original Dataset Schema Source: Microsoft

The version of the graph that we use was scrapped in November 2018. The complete graph has close to 209 million papers and 253 million papers.

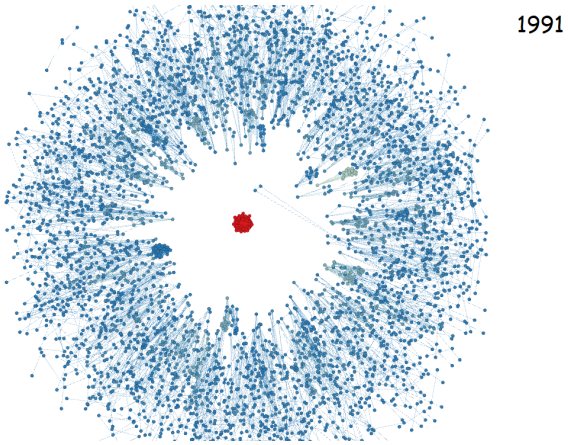

Fig. 3: Sample Co-Authorship Graph

D. *Dataset Preparation*

The raw RDF files contain unnecessary textual data. To clean the dataset and extract AuthorID, PaperID and Affiliations and Year of publication, Bash scripts are used. These bash scripts execute Regular expression filters to extract relevant data and written to respective CSV files. The grep command extracts the records which follow the given pattern and outputs them to a CSV file and because of the data contains unnecessary textual information, we clean it using sed command which simply replaces the redundant data with an empty string thereby giving us the required data.

Sample Bash scripts are shown below.

E. Data extraction

However, for this work, to generate our dataset we have used Authors, Papers and PaperAuthorAffiliations RDF files for the required years using python scripts.

F. Graph generation

The graph is formed with Authors as nodes and PaperAuthorAffiliations as edges. Authors file contains Author ID and Author Label, Papers file contains Paper ID, Paper Label and Year of publication, PaperAuthorAffiliations file contains the edge information between paper and authors. Since we want to form edges between author to author, PaperAuthorAffiliations have been reduced as an author to author affiliation by performing self-join with paper id as joining condition between authors so that authors will be the vertices and paper information will be added as edge labels. This will also make the edge the direction between authors as bidirectional, practically turning the directed graph to the undirected graph.

G. Train and Test data generation

From MAKG, we extract data for the latest 3 years. In order to partition the train and test data, uniform sampling from both the classes is performed. We first sampled the dataset where the link exists between two authors with the class label "Yes", which indicates that they have published a paper together. And then we sampled the same amount of author pairs where the link does not exist with the class label "No".So now we have an equal number of samples from both the classes. We performed a temporal split for generating train and test data. Graph data until a certain year is used for training purpose while the rest of it is used for testing.

H. Sample Dataset Generation

By performing random sampling of author pairs from the graph we observed huge class imbalance as our graph is skewed and most of the author pairs do not have links, to begin with. To maintain class balance, author pairs have been divided into two categories as a link exists and the link does not exist. We randomly pick an equal number of samples from each category to form our training and test data sets.

IV. Experimental Setup and Implementation

A. Dataset description

We have used the MAKG dataset of version 2018-11-09. We have selected graph data for the years 2016, 2017 and 2018 as our data set.

Years	Nodes	Edges
2016	4.5 Million	21 Million
2017	4.5 Million	7 Million
2018	4.5 Million	7 Million

TABLE I: Dataset Statistics

Training data set formed with labels from 2017 graph and features are calculated on the 2016 graph. Testing data set formed with labels from 2018 graph and features are calculated on 2016,17 graphs the directed graph to the undirected graph.

	Year	2016	2017	2018
Training Data	Features	X		
	Labels		X	
Test Data	Features	X	X	
	Labels			X

TABLE II: Feature and Table Selection

B. Software and Tools

In the following we list the tools and software that we have used to implement, for our study, a prototypical machine learning pipeline for graph data.

- **Apache Spark 2.3.0**- It is an In-Memory data processing engine [10] for large volumes of data with rich and concise high-level API's for Scala,Python,Java,R and SQL.
- **Apache-Spark MLlib 2.3.0**- MLlib [6] is Spark's machine learning library. It provides tools such as ML algorithms, VectorAssembler and utilities.
- **Apache-GraphX 2.3.0**- It is an API for parallel computation on graphs [9]. It provides an algorithm library for calculating features of graphs.
- **Apache-Spark GraphFrame 0.7.0**- It is a library which helps in converting graph data into Spark data frames called GraphFrames [3]. It provides tools for executing queries and standard graph algorithms on GraphFrames.
- **Gephi 0.9.2**- It is an open-source software [2] to analyze and visualize networks.
- **Neo4J 4.0**- It is a stand alone graph database management system [7] which uses Cypher query language [4] to execute queries and provides in-built graph algorithms.

C. Hardware Setup

- **For calculating features:** Our hadoop cluster is a HortonWorks data platform distribution with YARN as our cluster manager. We have conducted our series of experiments using Scala based Spark applications in our

cluster with Spark version 2.3. The cluster has ten nodes, which includes one edge node to access the cluster, one Active NameNode, one StandBy NameNode and seven DataNodes which the applications could use. The hardware specs of cluster are as follows:
NameNodes: 6 Cores, 32GB RAM, 150GB HardDisk
DataNodes: 4 Cores, 16GB RAM, 150GB HardDisk
Network: 1 GBit Ethernet in star topology.
The maximum allowable Yarn container size per node is 4 Cores, 10GB RAM. We submitted our applications as jar files to the cluster and conducted a series of experiments to evaluate the runtime. The features are calculated multiple times and the final result time is averaged.

- **For Link Prediction:** A 12GB Ram,64-bit operating system with i7-8th Gen processing at 1.8Ghz is used.
- **For Neo4j:** A machine with Ubuntu 18.04.3 LTS, and 72 Intel(R) Xeon(R) Gold 6150 CPU @ 2.70GHz processrs, with 395 GBs of memory.

D. Application Setup

SBT as packaging tool: Simple build tool (SBT) is used to package spark applications and its dependencies in our project.

*E. **Scaling down graph:** Our graph consists of 270 million vertices and 2.2 billion edges. Due to hardware limitations, we could not calculate the features for this dataset. Random sampling on the graph would lead to exclusion of dependent edges from the graph. To overcome this issue without losing any dependencies, BigStar and SmallStar algorithm from [5] is used. This algorithm alternately applies BigStar and SmallStar techniques to find the connected components in the graph until it converges. The algorithm outputs list of connected components which are then filtered out according to the size of the component to form scalable graphs.*

F. Feature Calculation

Offline Features: These features are precomputed for each vertex of the graph and are independent of author pairs. All the feature programs have main file driver for orchestration, utilities for handling the feature calculation tasks and constant file for the application parameters

- **Pagerank:** We made use of Pagerank API from spark and saved the results to a file. we pass the tolerance, reset-probability for dynamic Pagerank. Write the PageRank score against each vertex into csv files.
- **Preferential Attachment:** First we obtain the number of neighbours of all the authors in the graph using CollectNeighbours API. We normalize this with no of edges in the respective graph. Write the preferential attachment score against each vertex into csv files.

Online Features: These features depend on author pair.

- **Common Neighbours:** Vertices which are common between author pairs. First, we need to collect neighbours of all the vertices (authors) using CollectNeighbours API. Later we join them with the author tuples. We apply the intersection between the neighbours of the tuples that is we get the total number of common neighbours between the two authors. Finally, we get the length of common neighbours (count of common neighbours between author pair) and save it in a file.
- **First Shortest Path:** We extract the first value of the author tuple and then make it as a source. We use this as a landmark vertex and feed them to the ShortestPath API. Later we filter out the required source to target vertices from the author tuple and finally save the first shortest path to file.
- **Second Shortest Path:** We use Pregel API to calculate the second shortest path. First, we pass messages with hop counts to neighbouring vertices of the source. This message is updated over multiple iterations till it converges. Later we filter out the required source to target vertices from the author tuple and save the second shortest path to file.
- **Connected component:** It is calculated for each author pair as a binary feature - it is ”True”, if both authors belong to the same connected component, ”False” otherwise.

For the calculation of the features in Neo4j, we have used the implementations offered by the native Algorithms plugin.

G. Link Prediction Model

We used Pyspark and its machine learning library Spark MLlib to implement this model. We experimented for each combination of features and classifiers and logged its F1 score in MLFlow for easy tracking. In our experiments parameter settings for all classifiers have been set to default with respective to Spark MLlib.

V. Evaluation

1) **Feature Calculation**
Run time comparison between Spark and Neo4J:
We evaluated run times for features mentioned above for Spark and Neo4J using in both cases a high-end hardware setup that matches well the architectures. For

	Time Taken in seconds									
	V = 1.2M, E = 1.8M		V = 1.8M, E = 3.2M		V = 2.5M, E = 6M		V = 3.3M, E = 12M		V = 4.5M, E = 35M	
	S = 50K	S = 200K	S = 50K	S = 200K	S = 50K	S = 200K	S = 50K	S = 200K	S = 50K	S = 200K
Global Pagerank	150	150	170	170	175	175	190	190	310	310
Preferential Attachment	3	3	3	3	5	5	6	6	20	20
Common Neighbours	55	60	68	69	75	78	77	79	90	97
First Shortest Path	101	123	107	267	109	471	209	484	NA	NA
Second Shortest path	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Second Shortest path	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Connected Component	445	445	534	534	612	612	719	719	1434	1434

TABLE III: Feature run time on Spark, where V-vertices, E-edges, S- sample and M-million.

	Time Taken in seconds			
	E = 6M; V = 2.5 M		E = 12 M; V = 3.3 M	
	Sample size = 200,000			
	Spark	Neo4j	Spark	Neo4j
Global Pagerank	175	68	190	120
Preferential Attachment	5	610	6	1100
Common Neighbours	78	NA	79	NA
First Shortest Path	471	NA	484	NA
Second Shortest path	NA	NA	NA	NA

TABLE IV: Feature run time on Spark vs Neo4j, where V-vertices, E-edges, S- sample and M-million.

Neo4j we selected a high-end server, and for Spark we selected a cluster of 10 nodes. Below are our observations

Expectation: Spark is a high-performance general-purpose parallel computing framework with libraries available for graph, data processing. Hence, our expectation is that Spark can scale for huge graphs and perform much better than Neo4J on a high-end server.

Observation: From the results shown in TABLE II,

Pagerank: As observed Pagerank has a sub-linear increase in run time with an increase in the graph size.

Preferrential attachment: Since Preferential attachment is a local feature, it has the lowest run time and increases to a small extent with increase in the graph size.

Common neighbours: Common neighbours is completely an online local feature. Hence, it has the second-lowest run time and it is increasing sub linearly with increase in the graph size.

First shortest path: It is an online path based feature. Hence, the run time increases in an almost exponential manner with both graph size and samples.

Second shortest path: It is an online path based feature and it did not scale well for the considered graph.

Connected Component: It is an online community-based feature and the run time increased sub-linearly with an increase in the graph size.

we can also see that path based algorithms are surprisingly not scaling well to our expectations in spark and also did not scale on Neo4J. We can also see that for the features: Pagerank, Preferential Attachment and Common Neighbors, the spark is performing much better and scales easily for huge graphs. However we should also note that there are many cases, marked as NA, for which it was not possible to calculate the features, due to resource limitations of the frameworks evaluated.

Conclusion: Spark can work on huge graphs but Path based algorithms do not scale well. Even though Neo4J stores graph in its native data format and has inbuilt graph algorithms, it scaled only for Pagerank and Preferential attachment and not for path based features.

2) **Model Evaluation** We evaluated the performance of classical(Naive Bayes, Decision Tree) and ensemble (Random Forest and Gradient boosted tree) classifier models using F1 scores

Expectation: We expected ensemble models to perform better over the classical model. and expected naive bayes to perform the least.

Observation: From Table (Evaluation Metric Table for 35M edge graph), We observed the same behaviour as our expectation, this is in coherence with the findings cited in [11]. Random forest performed the best with an average F1 score: *0.64710*.

Conclusion: As observed Ensemble-based models performed much better than naive bayes model, with almost similar average F1 scores of *0.64710* and *0.64102* respectively.

3) **Feature Importance** We evaluated F1 scores for all combinations to find the best performing feature combination.

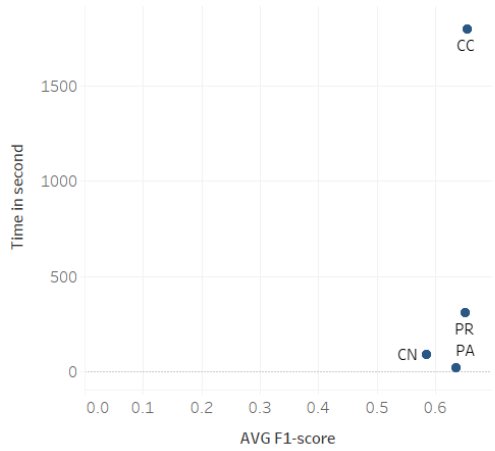

Fig. 4: Pareto front plot: Average execution time vs Average F1-score for features

Feature	NB	DT	GBT	RF
CC	0.33344	**0.73234**	**0.73234**	**0.73234**
CN	0.33344	**0.55667**	**0.55667**	**0.55667**
PA	0.44527	0.69577	**0.73392**	0.69292
PR	0.50278	**0.73095**	0.71237	0.72050
CN+CC	**0.55667**	**0.556677**	**0.55667**	**0.55667**
PA+CC	0.65421	0.70307	0.67682	**0.71462**
PA+CN	0.55666	0.55880	**0.59015**	0.56136
PR+CC	0.54373	0.74918	0.75990	***0.76990***
PR+CN	0.55388	0.60059	0.58235	**0.60072**
PR+PA	0.50278	0.63385	**0.71541**	0.63670
PA+CN+CC	0.55667	0.58577	**0.61393**	0.59241
PR+CN+CC	0.55387	0.62369	0.60992	**0.62811**
PR+PA+CC	0.54373	0.67086	**0.68901**	0.66299
PR+PA+CN	0.55388	0.57051	**0.58311**	0.58502
PR+PA+CN+CC	0.55387	0.59553	0.59399	**0.60434**

TABLE V: Evaluation Metric Table for 35M edge graph

Expectation: We expected Common neighbours and path based features to contribute more for link Prediction. Because two authors having common co-author are more likely to collaborate in future, Two authors can also be connected indirectly through other authors. Hence, they are more likely to share similar interests which might depend on path length.

Observation: From Table (Evaluation Metric Table for 35M edge graph), We can observe that Pagerank, connected components and Preferential Attachment features are contributing more, compared to other features, and it is also observed that calculating these features is cheaper than calculating path-based features in Spark.

Cost benefit analysis: Fig 3 represents a Pareto plot where the line describes the average F1 score for each feature and bar chart represents the time to take to calculate each feature. From this, we can deduce that, even though Pagerank and preferential attachment have slightly less F1 scores than the connected components, they are relatively cheaper to compute. Hence, in cases where we need to conserve resources just using Pagerank or Preferential attachment would suffice for link Prediction.

Conclusion: Among all the combinations, the Pagerank with Connected component resulted in the highest F1 score - *0.7699* on Random Forest classifier for a graph with 4.5 Million vertices and 35 Million edges. Since the graph is extremely sparse with a very small number of hugely connected components, and a very large number of small connected components, it is seen that Common neighbours and path based algorithms did not contribute well towards link prediction. But community-based features were able to successfully classify more author pairs compared to other features.

VI. Related Work:

In this section we will discuss other work that has been done in this domain, for link prediction.

A detailed survey on link prediction has been conducted by Victor et. al. [11]. The experiment is conducted on seven different datasets on various link prediction approaches. According to their observation, local features of the graph contribute more towards link prediction while indirect links of the graph increased the computational complexity and having no significant contribution in achieving the goal. However, this is contradictory to the conclusions made from our experiment that community-based features have a higher contribution. This might be attributed to the disparate nature of our dataset, where there are many locally connected small components and not densely connected.

Secondly, we can take the survey on Link prediction using in supervised learning by Al Hasan et. al. [1]. Here, they made use of different types of features set to do link prediction(co-authorship) between two authors. They have made use of non-topological features such as proximity text features which take keyword match count into consideration. They also used aggregated features which include the sum of papers, the sum of neighbours, the sum of keyword count, the sum of classification node and the sum of the log. At last, they have, made use of topological features like shortest distance, clustering index and shortest distance in author-KW graph.

Different classification model like decision tree, k-nn, multi-layer perceptron, SVM, rbf network was used to predict the link. Among them, SVM showed the highest accuracy. They have taken two datasets(BIOBASE and DBLP). The shortest distance KW-author graph feature contributed more towards the link prediction BIOBASE database whilst it was the sum of paper for DBLP. Whereas, as in our experiment, we took MKG datasets for which random forest showed the highest accuracy and Pagerank and Preferential attachment contributed more towards the link prediction.

VII. Conclusion:

In this research, we have experimented on feature calculation on graphs and samples of various size's using Spark and Neo4j, on both tools. Path based features did not scale well as the graph size increased. Community-based features such as Preferential attachment and common neighbours were cheapest to compute. We have experimented with combinations of features, models and sample sizes, community-based features and Ensemble-based models both had overall highest average F1 score and also highest F1 score as compared to all the other features. We also observed that change in sample size had a negligible effect on the final F1 scores.

VIII. Future work:

Data Perspective: We could not able to analyze the complete graph for more than three years due to limitations of the cluster. We considered only the co-authorship network in the current setup. IN future we could also combine with Citation network, Conference network and more for better link prediction.

Feature Perspective: In this current setup, we did not consider graph embeddings (e.g. entity embeddings), non topological features such as common Area of interests, in-paper keyword match, apart from other topological features.

Model Perspective: In this paper we have used traditional classifier models, for which features are important. However, for novel deep learning models, features might be less important as they are able to learn over raw graph data (e.g. with graph convolutional neural networks). This constitutes a possible area for continuing our work.

Model Perspective: In this work we studied a large-scale graph processing framework, and a graph database. In future work alternative choices for these tools could be considered.

References

[1] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. In *SDM06: workshop on link analysis, counter-terrorism and security*, volume 30, pages 798–805, 2006.

[2] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: an open source software for exploring and manipulating networks. In *Third international AAAI conference on weblogs and social media*, 2009.

[3] Ankur Dave, Alekh Jindal, Li Erran Li, Reynold Xin, Joseph Gonzalez, and Matei Zaharia. Graphframes: an integrated api for mixing graph and relational queries. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, pages 1–8, 2016.

[4] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1433–1445, 2018.

[5] Raimondas Kiveris, Silvio Lattanzi, Vahab Mirrokni, Vibhor Rastogi, and Sergei Vassilvitskii. Connected components in mapreduce and beyond. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–13, 2014.

[6] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[7] Mark Needham and Amy E Hodler. *Graph Algorithms: Practical Examples in Apache Spark and Neo4j*. O'Reilly Media, 2019.

[8] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Hsu, and Kuansan Wang. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*, pages 243–246, 2015.

[9] Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, and Ion Stoica. Graphx: A resilient distributed graph system on spark. In *First international workshop on graph data management experiences and systems*, pages 1–6, 2013.

[10] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[11] Zhengzhong Zeng, Ke-Jia Chen, Shaobo Zhang, and Haijin Zhang. A link prediction approach using semi-supervised learning in dynamic networks. In *2013 Sixth International Conference on Advanced Computational Intelligence (ICACI)*, pages 276–280. IEEE, 2013.

APPENDIX

We also experimented graphs with sizes of 6M and 12M edges and we observed that the highest F1 scores were much lesser than the one we observed with 35M graph. Hence, like the no of edges increases, the ability to predict links also increases. We could also see that First shortest path itself did not contribute much. But along with connected components, we were able to achieve an F1 score of 0.7095 for Random forest and Gradient boosted tree.

Features	NB	DT	GBT	RF
CC	0.39032	**0.71076**	**0.71076**	**0.71076**
CN	0.39032	0.39032	0.39032	0.39032
FSP	0.39032	0.39032	0.39032	0.39032
PA	0.39032	**0.64649**	0.63770	**0.64649**
PR	0.44300	**0.61294**	0.58391	0.61057
CC+CN	0.67783	0.67783	0.67783	0.67783
CC+FSP	0.39032	**0.70905**	**0.70905**	**0.70905**
CC+PA	0.39032	**0.64237**	0.60705	**0.64237**
CC+PR	0.62420	**0.68530**	0.66952	0.67846
CN+FSP	**0.41278**	0.39032	0.39032	0.39032
CN+PA	0.39032	0.64500	0.61447	**0.65346**
CN+PR	0.41740	0.50524	0.51192	**0.51346**
FSP+PA	0.39032	0.64649	0.64517	**0.66536**
FSP+PR	0.43957	0.61294	0.58586	**0.61390**
PA+PR	0.44301	0.58672	**0.60377**	0.59420
PA+CN+CC	**0.67783**	0.66827	0.63093	0.67640
CN+FSP+CC	0.67623	**0.67740**	**0.67740**	**0.67740**
PA+CN+FSP	0.40907	0.64500	0.62604	**0.66829**
PA+FSP+CC	0.39032	**0.64237**	0.60434	0.64078
PR+CN+CC	0.41736	0.60441	**0.61976**	0.63401
PR+CN+FSP	0.41673	0.50524	0.51147	0.50511
PR+FSP+CC	0.62364	0.68530	0.66309	**0.68697**
PR+PA+CC	0.62420	**0.63266**	0.62205	0.61271
PR+PA+CN	0.41740	0.53263	**0.56743**	0.53298
PR+PA+FSP	0.43957	0.58672	**0.60293**	0.58268
PA+CN+FSP+CC	0.67623	0.66827	0.63404	**0.67655**
PR+CN+FSP+CC	0.41700	0.60441	0.62159	**0.63099**
PR+PA+CN+CC	0.41736	0.62247	0.61463	**0.62759**
PR+PA+CN+FSP	0.41673	0.53263	**0.56450**	0.52259
PR+PA+FSP+CC	0.62364	**0.63266**	0.62236	0.61875
PR+PA+CN+FSP+CC	0.41700	0.62247	0.61577	**0.63324**

TABLE VI: Evaluation Metric Table for 12M edge graph