NUS | Computing

National University of Singapore

CS5425 Big Data Systems for Data Science

Project Report

Analysis of Large Scale Academic Social Networks

Submitted by: Group 23

Name	**Matriculation Number**
Cai Yufan	A0219252J
Chong Sai King	A0194964L
Du Zidong	A0194934R
Huang Xuankun	A0185827N
Khoo Ju Seng Bryant	A0198804R

Date: 28-Nov-20

1. Project Introduction

This project has provided a comprehensive analysis of the latest academic social networks. The main tasks performed include graph exploratory analysis, research topic analysis, self-citation analysis, community detection and link prediction. The research topic analysis identifies the historical trend of popular topics over the past couple of years and which fields of study are booming with activity. The self-citation analysis has discovered authors and groups who always cite their own work. The community detection identifies the communities, understands what it takes for certain communities to be grouped together and models groups of people with similar interests. The link prediction has experimented with both similarity and learning-based approaches and the prediction is beneficial in providing potential academics with an understanding of who to best collaborate with, promoting communication between them. Algorithms for Social Network Analysis (SNA) are usually complex that makes it difficult to implement in big data environments on large network datasets (Soric et al., 2017). In this project, the various SNA algorithms are validated in academic social networks and implemented using Apache Spark, a distributed general-purpose cluster-computing framework. Lastly, an interactive Power BI dashboard (click here to access) is developed to automatically load and visualize the data generated by various algorithms which has formed an end-to-end data analysis process.

2. Methodology and Experimentation

2.1 Dataset and Tools

The AMiner citation dataset (Tang et al., 2008) has been chosen in this project. The citation data is extracted from DBLP, ACM, MAG (Microsoft Academic Graph) and other sources. The latest snapshot of the dataset as of 05-May-19 is selected which includes 4,107,340 papers with a size of 11.2G in nested JSON format. The data is processed and split into the following four sub-datasets. The paper data has 4,107,340 papers, the author data has 3,655,052 authors, the paper and author relationship data has 12,407,114 paper and author relationships and the co-author data has 11,697,041 co-authorships. The tools used for this project include Azure Databricks, which is an Apache Spark based analytics service, Azure Blob Storage for data storage and Power BI for data visualization. The programming languages used are Python and Scala. The Databricks cluster is configured with autoscaling of 2 to 8 workers and each worker has 28GB memory, 8 Cores and 1.5 DBU. The driver has 28GB memory, 8 Cores and 1.5 DBU.

2.2 Analysis of Research Topics

The trend analysis portion of this project aims to study the changes in popularity in research topics over the years based on the keywords and abstract of the papers. For this part of the project, we define popularity of a research topic based on the frequency of the topics. In subsequent sections, we will then explore the influence at the node level of the dataset. We will take a text mining approach for this part of the project. The relevant fields in the dataset that would be considered during our experiment would be text fields such as title, abstract, keywords and fields of study. We will perform this part by splitting documents based on publication date.

2.2.1 Topic Modelling with Latent Dirichlet Allocation (LDA)

Since the documents are not evenly distributed, we split up the dataset into various data frames by the following time periods. Years 1950 to 1979, years 1980 to 1989, years 1990 to 1999, years 2000 to 2009, years 2010 to 2013, years 2014 to 2016 and years 2017 to 2020. The following pipeline shown in Figure 2.2.1 demonstrates how we perform text processing before running the LDA model. We firstly tokenize both the “title” and “keywords” field in the dataframes. Then the stop words in each document are removed before these two columns are combined together. The output column then has its term frequency counted before the term frequency inverse document frequency is calculated for each document. The formula for a term i in document j is $w_{i,j} = tf_{i,j} * log(\frac{N}{df_i})$, where $tf_{i,j}$ is the term frequency referring to the number of occurrences of i in j, df_i is the number of documents containing i, and N is the total number documents.

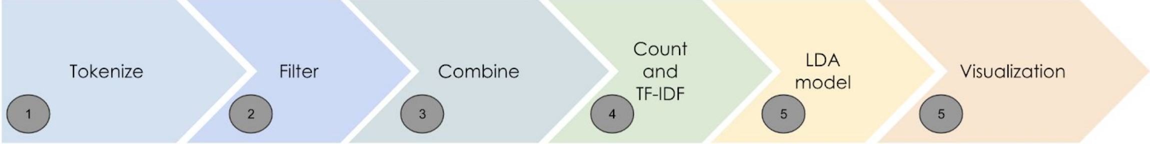


Figure 2.2.1 Pipeline for LDA

Once the text processing steps have been performed, the most important step in our pipeline is running Latent Dirichlet Allocation. The algorithm groups the nodes into topics using Latent Dirichlet Allocation (LDA). LDA works under the underlying assumption that similar topics make use of similar words and that the statistical distributions of topics can be determined. We use LDA to map documents in our corpus to topics that can cover the words in our dataset. Initially, LDA was run with 10 topics, and the resulting topic clusters were overlapping and too near to one another. The process of assigning a better topic number was empirical and we decided to go with 5 topics in the end. The resulting clusters were then well defined and had clear differences in meanings. How LDA works is that it temporarily assigns each word to a topic via dirichlet distribution. If two words happen to be in the same document, there is a possibility of each word being assigned to a different document. Lastly, an iterative process will check and update topic assignments based on how prevalent a word is in the document and how prevalent are topics in the document.The final output of LDA is the topic assigned to each word in the document. What we can then interpret from the result is determining the meaning of each topic, based on the top most occurring word in each cluster.

2.2.2 Implementation using Big Data Systems for Topic Modelling

As we are working with a relatively big dataset, steps 1 to 5 in our pipeline described above are implemented using pyspark. Using the pyspark library allows us to overcome the main issues of working on large datasets which are the large amounts of I/Os in hard disk and network, as well as large memory requirements for each spark job. Running the pipeline with spark on the databricks cluster was essential in this case. Once we had the topic clusters, we used a simple visualisation library called pyLDAvis for visualising the topic clusters. The visualisations are also included in our Power BI dashboards.

2.3 Identification of Influencers

The chosen citation dataset contains information of paper collaborations, which uncovers a huge amount of interpersonal relations in academia. It makes influencer ranking possible by detecting and ranking influential scholars. We intend to make use of the connection relations of scholars exposed in the dataset, to identify not only general influencers in academia, but also domain-specific opinion leaders in specific fields of study.

2.3.1 Data Preparation & Processing

The original data set is formatted in paper-level, and being processed into 3 dataset: Papers, Authors and Co-Authorship. In order to generate a more comprehensive author-level data set for influencing author identification, further processing is done to aggregate the following author-level informations: (1) Paper count and total citations; (2) Degree-centrality; (3) PageRank weight; (4) Self-citation rate (elaborate in Section 3.3) Degree centrality, which represents the connectivity of each node, is being measured using the collaboration network between authors by aggregating the unique collaboration edges between authors.

2.3.2 PageRank Implementation for Influencer Identification

Based on the similarity of citation networks and internet node characteristics in structure, various network ranking algorithms can also be modified and implemented to evaluate the ranking of influencers in academia. In this case, PageRank algorithm is chosen to identify: (1) influencing authors in co-authorship level with 3.6M nodes and 11M edges; (2) influencing papers in co-citationship level with 10M nodes and 300M edges.

The implementation of PageRank algorithm is done using Graph API form the GraphFrame library, with teleport probability being set as 0.15, and max-iteration being set to 20. The reason for choosing max-iteration instead of tolerance as the limiting factors is due to the huge amount of edges in our graphs, where computation of a single iteration on co-authorship level is approximately 12 mins, and on co-citationship level is 2 hours. The usage of PageRank results will be elaborated in Section 3.4 Community Detection.

2.4 Self-Citation Detection and Analysis

Self-citation rate detection aims to detect authors with high self-citations, and to analyze the similarity of the author community based on their self-citation rate. Self-citation rate is calculated by number of self-citations over the total citation received. Citations from all papers of an author are extracted, with sum of citations from self-written papers being aggregated.

2.4.1 Self-Citation Analysis Based on Topic Detection & Venue Comparison

To get further insights from authors self-citation rates, author data are being processed by their self-citation rate. Each set of data is being further processed by extracting the relevant papers of each node, with focus on the paper's venue, title and research keywords. For topic detection analysis, keywords and title of each paper are being extracted into key-term data set, and processed with topic detection model with LDA. The topic clusters from each SCR data set consists of the frequency of most common research key terms. The largest topic clusters will be selected and compared with their research key terms for observation of their research topic/trends. For venue topic analysis, papers of each set of authors with different SCR, are grouped by their corresponding venue. The top venues with the largest number of research papers from different SCR sets are being compared.

2.5 Community Detection

The community detection in this project aims to detect the communities through analyzing the links (collaborations) between the nodes (authors). There are 3 different community detection algorithms experimented in this section.

2.5.1 Louvain Algorithm

The Louvain is a greedy agglomerative hierarchical algorithm that forms communities by maximising the modularity of the network (Blondel et al., 2008). There are two phases in each iteration. First Phase (Modularity Optimization): Firstly, each node is assigned to its own community. Then, it uses local moving heuristic to obtain a better community structure by placing the nodes into the community that resulted in the greatest modularity increase. This process repeats until no further modularity increment. Second Phase (Community Aggregation): In the second phase, all the nodes that belong to the same community are grouped together to form a new network. The intra-community links are represented by self-loops, whereas the inter-community links are aggregated and represented as edges between the communities. This process repeats until only one community remains. This algorithm is implemented using the *"community.community_louvain.best_partition"* API from networkx library.

2.5.2 Label Propagation Algorithm (LPA)

Label Propagation Algorithm (LPA) is an iterative community detection algorithm that forms communities by propagating the labels throughout the network via underlying edge structure (Raghavan et al., 2007). In this algorithm, all densely connected nodes will be quickly assigned to a unique community label in just a few iterations. Eventually, most of the labels will be discarded and only few labels will remain. The algorithm works as follows: (1) Initialize each node with a unique community label. (2) For each iteration, propagate all the nodes randomly. (3) For each node, update its label with the label of the majority of its neighbours. Ties are broken uniformly and randomly. (4) LPA stops when each node is labelled with the majority label of its neighbours or user-defined maximum number of iterations is achieved.

This algorithm is implemented using the *"graphframes.GraphFrame.labelPropagation"* API from Spark GraphFrames.

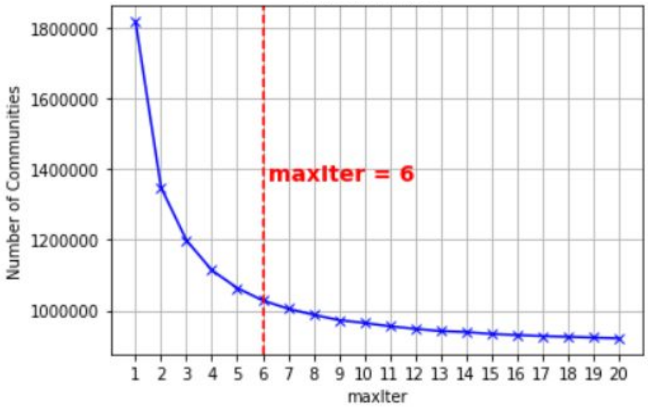


Figure 2.5.1 Elbow Method

In this project, Elbow method is used to find the optimal number of max iterations k for LPA. The intuition is, the "elbow" point will act as a cutoff point to select the optimal number of iterations where adding more iterations doesn't give much better clustering performance. Based on the line chart above, the optimal number of iterations is 6. Hence, we select maxIter=6 as our hyperparameter.

2.5.3 Connected Components (CC)

Connected Components (CC) forms clusters by finding sets of connected nodes in an undirected graph where each node is reachable from any other nodes via network edges (Levorato & Petermann, 2011). The algorithm works as follows: (1) Initialize all nodes as "not visited". (2) For every node v, if node is "not visited", mark node as "visited", and add it to its respective set of connected nodes. (3) For every adjacent node u of v, if u is "not visited", repeat Step 2 recursively. (4) Label each set of connected nodes with a unique community label after all nodes are marked as "visited". This algorithm is implemented using the *"graphframes.GraphFrame.connectedComponents"* API from Spark GraphFrames.

The results and analysis of all three algorithms will be discussed in Section 3.3

2.6 Link Prediction

The link prediction aims to predict the new collaboration between the authors for a future time t' (t' > t) given the academic social network G(V, E) and a particular time t, where V and E are the sets of nodes (authors) and links (collaborations) respectively. The generic link prediction framework shown in Figure 2.6.1 has been adopted and both similarity-based and learning-based approaches for link prediction have been experimented.

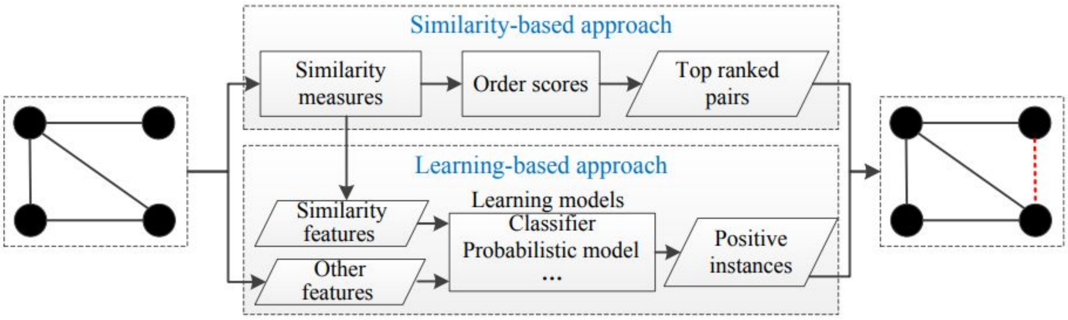


Figure 2.6.1 The Generic Link Prediction Framework (Wang et al., 2015)

2.6.1 Data Preparation for Link Prediction

<table>
<tr><td>Train</td><td colspan="3">Feature Network</td><td colspan="3">Label Network</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Test</td><td colspan="6">Feature Network</td><td colspan="4">Label Network</td></tr>
<tr><td>Year</td><td>1990</td><td>......</td><td>2012</td><td>2013</td><td>2014</td><td>2015</td><td>2016</td><td>2017</td><td>2018</td><td>2019</td></tr>
</table>

Figure 2.6.2 The data preparation for link prediction

The coauthor network has been partitioned into two two non-overlapping periods (i.e. a feature network and a label network) based on the first year of collaboration between the authors (Figure 2.6.2). To align with the temporal definition of link prediction, the train data has chosen 2012 as the time to split feature and label network and the test data has chosen 2015 as the time to split the feature and label network. In short, we use features based on the network as of 2012 and observe whether they collaborate in the next three years from 2013 to 2015. The test data is to use the new features derived based on the network as of 2015 and predict whether they will collaborate in the next period from 2016 to 2019. The candidate author paris are formed based on the $\frac{N(N-1)}{2}$ rule by choosing authors who publish papers during both the feature network and label network periods. If the author pairs have already collaborated in the feature network, they will be excluded from the candidate author pairs. The candidate author paris are labelled as 1 if the author pair has collaborated in the label network and otherwise will be 0 (Needham & Hodler, n.d.). The dataset derived is very imbalanced and the methods to be adopted to address the issue include undersampling negative pairs or dropping negative pairs arbitrarily by choosing pairs that are N-hops away (Lichtenwalter et al., 2010). In this project, the undersampling technique is used to balance the dataset. The train dataset has 522,718 (1: 260,557, 0: 262,161) records and the test data has 477,786 (1: 237,884, 0: 239,902) records. Lastly, the features are derived for each labeled author pair based on the feature network.

AUC (Area under the ROC Curve) score is used to evaluate the model. The threshold to determine the prediction of 1 or 0 for various measures is different. AUC curve is generated based on TPR/FPR of different thresholds which enables us to evaluate models independently of the threshold because it only considers the rank of each prediction instead of its absolute value. AUC is also a robust evaluation criteria in presence of imbalance.

2.6.2 Similarity-based Approach

A similarity-based approach computes the similarities on non-connected pairs of nodes. It is one of the most common approaches used for link prediction especially for large social networks. The following 11 similarity metrics have been adopted to perform link prediction and the metrics computation is implemented on Spark using Scala.

Metrics	Description	Formula
Common Neighbors (CN)	CN computes the number of common connections between the two nodes. The idea is two strangers who have connections in common are more likely to be connected than those who do not have connections in common.	$CN(x,y) = \lvert N(x) \cap N(y) \rvert$
Total Neighbors (TN)	TN computes the number of total connections, being common or not, between the two nodes. The idea is the more neighbors they have, the more likely they will be connected.	$TN(x,y) = \lvert N(x) \cup N(y) \rvert$
Preferential Attachment (PA)	The idea of PA is that the more connected a node is, the more likely it is to receive new links.	$PA(x,y) = \lvert N(x) \rvert \times \lvert N(y) \rvert$
Jaccard Coefficient (JC)	The Jaccard Coefficient assumes that two nodes which have a higher proportion of common neighbors relative to the total neighbors tend to connect.	$JC(x,y) = \frac{\lvert N(x) \cap N(y) \rvert}{\lvert N(x) \cup N(y) \rvert}$
Adamic Adar (AA)	The AA metric suppresses the contribution of the high-degree common neighbors. A value of 0 indicates that two nodes are not close while higher values indicate nodes are closer.	$AA(x,y) = \sum_{u \varepsilon N(x) \cap N(y)} \frac{1}{log \lvert N(u) \rvert}$
Resource Allocation (RA)	RA has a similar form like AA. However, it punishes the high-degree common neighbors more heavily than AA. Also, a value of 0 indicates that two nodes are not close while higher values indicate nodes are closer.	$RA(x,y) = \sum_{u \varepsilon N(x) \cap N(y)} \frac{1}{\lvert N(u) \rvert}$
Leicht Holme Nerman (LHN)	The idea of LHN is to assign high similarity to node pairs which have more common neighbors compared to the expected number of such neighbors. The higher the value, the closer the nodes are.	$LHN(x,y) = \frac{\lvert N(x) \cap N(y) \rvert}{\lvert N(x) \rvert \times \lvert N(y) \rvert}$
Sorensen Index (SI)	SI assigns high similarity to node pairs which have more common neighbors compared to the sum of the number of neighbors that the node pairs have. The higher the value, the closer the nodes are.	$SI(x,y) = \frac{\lvert N(x) \cap N(y) \rvert}{\lvert N(x) \rvert + \lvert N(y) \rvert}$
Salton Cosine Similarity (SC)	SC assigns high similarity to node pairs which have more common neighbors compared to the square root of the expected number of such neighbors that the node pairs have. The higher the value, the closer the nodes are.	$SC(x,y) = \frac{\lvert N(x) \cap N(y) \rvert}{\sqrt{\lvert N(x) \rvert \times \lvert N(y) \rvert}}$
Hub Promoted (HP)	HP is the common neighbors divided by the lower degree of nodes from the node paris.	$HP(x,y) = \frac{\lvert N(x) \cap N(y) \rvert}{min(\lvert N(x) \rvert, \lvert N(y) \rvert)}$
Hub Depressed (HD)	HP is the common neighbors divided by the higher degree of nodes from the node paris.	$HD(x,y) = \frac{\lvert N(x) \cap N(y) \rvert}{max(\lvert N(x) \rvert, \lvert N(y) \rvert)}$

Table 2.6.3 Similarity Metrics

2.6.3 Learning-based Approach

The feature-based supervised learning approach, which converts link prediction tasks to a binary classification problem, has been experimented. The Spark ML package is used and the classifiers selected to perform the classification task include Logistic Regression, Support Vector Machine, Random Forest, Gradient Boosted Trees and XGBoost. The train data is further split into 70% train and 30% validation. Grid search is used to fine-tune the parameters for classifiers with 5 folds cross validation. The BinaryClassificationEvaluator from Spark ML is used for evaluation metrics and its default metric is AUC.

Category	Features
Neighbor-based	Common Neighbors, Total Neighbors, Preferential Attachment, Jaccard Coefficient, Adamic Adar, Resource Allocation, Leicht Holme Nerman, Sorensen Index, Salton Cosine Similarity, Hub Promoted, Hub Depressed
Community-based	Triangle Count, Label Propagation, Connected Components
Node-based	Node Embeddings, Cosine Similarity of Research Interests

Table 2.6.4 Features for Supervised Learning

Node2Vec (Grover & Leskovec, 2016) algorithm is selected to generate the node embeddings. Node2Vec is a popular algorithmic framework for learning feature representations for nodes in networks. One of the limitations of the DeepWalk (Perozzi et al., 2014) algorithm is that the path cannot be controlled as it walks randomly. The main difference between Node2Vec and DeepWalk is instead of walking randomly, Node2Vec introduced the Breadth-Fast-Sampling (BFS) and Depth-First-Sampling (DFS) to control the random behavior. BFS reaches immediate neighbors while DFS prefers nodes away from the source. The parameter p prioritizes a BFS procedure, while the parameter q prioritizes a DFS procedure. The walk bias variable α is parameterized by p and q.

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

The Node2Vec algorithm is implemented using Fugue, which is an abstraction layer on top of different frameworks such as Spark and Dask etc. It aims to unify different computing and machine learning frameworks into one ecosystem (*Databricks*, 2020). There are three steps in the Node2Vec implementation which are indexing of the graph, sampling of random walks and generation of node embeddings. The output of Node2Vec is a 64-dimension vector for each node and the vector for each pair of nodes are concatenated to form a 128-dimension feature vector. It is then used together with the neighbor and community-based features.

Overall, three experiments have been set up for link prediction: (1) Use neighbor-based features to train link prediction models (11 features); (2) Use neighbor and community-based features to train link prediction models (14 features); (3) Use neighbor, community and node-based features to train link prediction models (143 features)

2.6.4 Graph Neural Networks

Recently, a new neural network called graph neural network has been put forward. It shows state-of-art performance in many tasks including computer vision and natural language processing. A GNN usually consists of 1) graph convolution layers which extract local substructure features for individual nodes, and 2) a graph aggregation layer which aggregates node-level features into a graph-level feature vector. It uses the graph neural network to learn a function mapping the graph features to link existence.

Model Architecture

In this part, we first introduce graph convolutional networks(GCNs), scalable GCN and then introduce an architecture that uses GCNs at its core for link prediction. The basic graph convolutional network (Kipf and Welling, 2017) is an adaptation of the convolutional neural network (LeCun et al., 1998) for encoding graphs. Given a graph with n nodes, we represent the graph with an $n \times n$ adjacency matrix A where $A_{ij} = 1$ if there is an edge between node i and j. In an L-layer GCN, we compute the h^l layer by following graph convolution operation:

$$h_i^l = \sigma(\sum_{j=1}^{n} A_{ij} W^l h_j^{l-1} + b^l)$$

Where h_i^{l-1} denote the input vector, h_i^l denote the output vector of node i at the l-th layer, W^l is a linear transformation, b^l is a bias term and σ is a nonlinear function (like ReLU). In that case, each node gathers the information from its neighboring nodes in the graph during the graph convolution.

We now adapt the graph convolution operation to an academic social network. We regard each link in the social network as a node v in the graph. The feature of the node is the original feature and artificial features like common neighbors, total neighbors, preferential attachment, jaccard, adamic adar, resource allocation, leicht holme nerman, sorensen index, salton cosine similarity, hub_promoted, hub_depressed. We regard the node with the common author id as linked. In that formula, the similar link in the academic social network will be clustered as they should have similar features. The neighbors of each node in the new graph will help further predict the label of the node because their neighbors are similar nodes linked by related authors.

Further Research

We also try to do some research on graph neural networks. For the basic GCN, it is simple and easy to realize but needs to train on the whole graph dataset. This would be difficult for a large scale dataset. Then I research the Graphsage, its main idea is to sample the fixed number of neighbours for each node. In that case, although it would lose some information of neighbours, it would be much faster and can be trained by minibatch. But we could find that if we run multiple layers of GCN, the cost will be exponentially related to the number of layers of GCN. If we want more neighbours for each node, the calculation would be much bigger. There are many ways to speed up the training process like FastGCN. It uses not only fixed numbers but also fixed neighbours. This method would reduce exponential calculation time to linear level. However, for large-scale graphs, sharing the neighbor set will make the adjacent vertices in the mini-batch too sparse, which will affect the convergence of the model. As we find that there are many neighborhoods that would be used multiple times when the training gets deeper layers, it would be a good idea to store the lower embedding layers like the cache mechanism in the spark.

We try to find a way to store some intermediate results in order to reduce the calculation. After looking at what have been experimented in the past, I find alibaba puts forward a new way called scalable GCN. For detail, considering the low-order embedding of adjacent vertices is repeatedly referenced between mini-batches, it introduces embedding cache, directly query the embedding of adjacent vertices (like the red nodes in the figure) and update the center vertex embedding(blue in the figure).

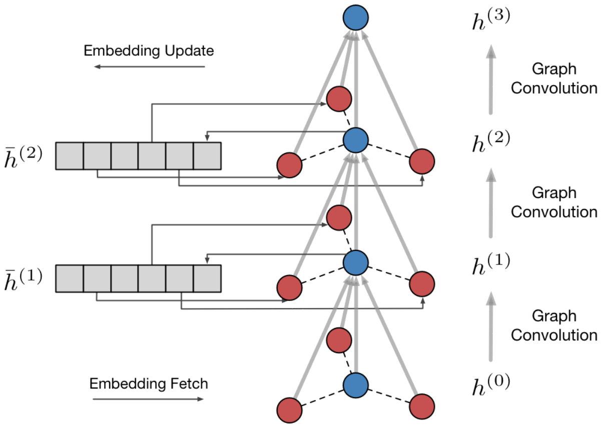


Figure 2.6.5 The model framework for multi-layer GCN (https://github.com/alibaba/euler)

2.6.5 Distributed Training

Although spark supports traditional machine learning algorithms like xgboost and random forest, it still does not support deep learning architecture like neural networks. Nowadays, neural networks show very strong power on classification problems. It is necessary and important to have some research in that part for further and deep understanding of the future of the spark.

Here we focus on the pytorch framework and research the distributed training algorithm including “data parallel”, “distributed data parallel” and “horovod”. The parallel can be classified to two types: model parallelism and data parallelism. Model parallelism means that the network is too large to store on one core, then split, and perform model parallel training. Data parallelism means that multiple GPUs simultaneously use data to train a copy of the network. Data parallel operation requires us to divide the data into multiple pieces and then send them to multiple GPUs for parallel calculations. Multi-core training requires communication overhead to be considered. It is a trade off process. It does not necessarily mean that four cores will be much faster than two cores as it may be that the communication overhead has already accounted for the bulk of the training for four cores.

In the torch, the method is to use several functions in nn.parallel. The functions implemented respectively are as follows: Replicate: Copy the model to multiple GPUs; Distribution (Scatter): Divide the input data into multiple copies according to its first dimension (usually the batch size), and transmit them to multiple GPUs; Gather: The data sent back from multiple GPUs are connected back together again; Parallel application (parallel_apply): Apply the distributed input data obtained in the third step to the multiple models copied in the first step.

The package data parallel separates one batch of data into some pieces and calculates each on a different GPU. The gradient will be summarized to the GPU zero and after back propagation on GPU zero it broadcasts the new parameters to all other GPUs. The problem is that GPU zero would be overloaded as it processes the back propagation alone. For package DistributedDataParallel, the GPU zero receives all the gradients and computes the average then only broadcasts the average gradient. Every GPU has its own optimizer so less data transfered. In that case, the load will be more balanced and the performance will be more efficient.

Finally, “Hovorod” is an industry product developed by Uber. It supports many popular machine learning platforms including Tensorflow, PyTorch, MXNet, Keras. It is a distributed training framework that supports data parallelism, synchronous update gradients and AllReduce collective communication. The AllReduce process integrates many communication frameworks including: NCCL(NVIDIA Collective Communications Library), MPI. NCCL is a library that can realize aggregate communication between multiple GPUs and multiple nodes. It is highly optimized and compatible with MPI, and can perceive GPU topology, promote multi-GPU multi-node acceleration, and maximize bandwidth utilization within the GPU. Therefore, researchers of deep learning frameworks can take advantage of NCCL's advantages at multiple nodes.

3. Discussion of Results

3.1 Analysis of Research Topics

The following insights were derived from the In the earlier years (1950s to 1980s) the research topics mainly revolved around fundamental research for Computer Science. The topic clusters mainly included generic words such as Mathematics and Boolean logic. This was a time when pioneers in the field started laying the groundwork for future researchers. In the 1990s, one emerging topic cluster contained terms such as “architectures”, “distributed” and “parallel”. We infer from this cluster that in the 1990s research in more advanced areas such as distributed computing were the trend. In the 2000s, the most salient terms were “web”, “network” and “wireless”. This is probably due to the internet boom, which led to massive resources invested on the internet. From 2014 to 2020, a new AI topic cluster started to emerge, which is no surprise. The salient terms for those topics were terms like “artificial”, “recognition”, “optimization” and “learning”. The topics that we observe so far make sense and give us greater insight on how research topics have evolved over the decades.

3.2 Self-Citation Detection and Analysis

A self-citation detection system to detect authors is built by aggregating author citations, papercount and self-citation rate, and group authors based on self-citation rate. The detail can be visualized in the Dashboard.

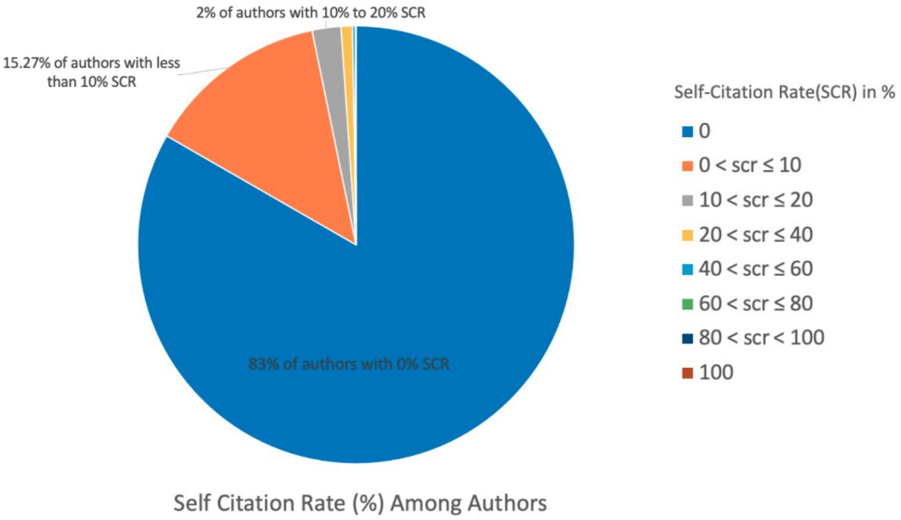


Figure 3.2.1: Distribution of author's self-citation rate

The distribution of SCR is shown in the following figure above, where the majority of authors (83.5%) have 0% SCR, and 17.3% of the authors have less than 20% SCR. This implies most authors have extremely low, or to say 0% SCR. More in-depth analysis is then performed by exploring authors' citations, papercount and research topics/areas.

3.2.1 Self-Citation Rate vs Research Workdone

To obtain more insights from the author's SCR, we further explore the relationship between SCR and research work done based on author's citations and paper count.

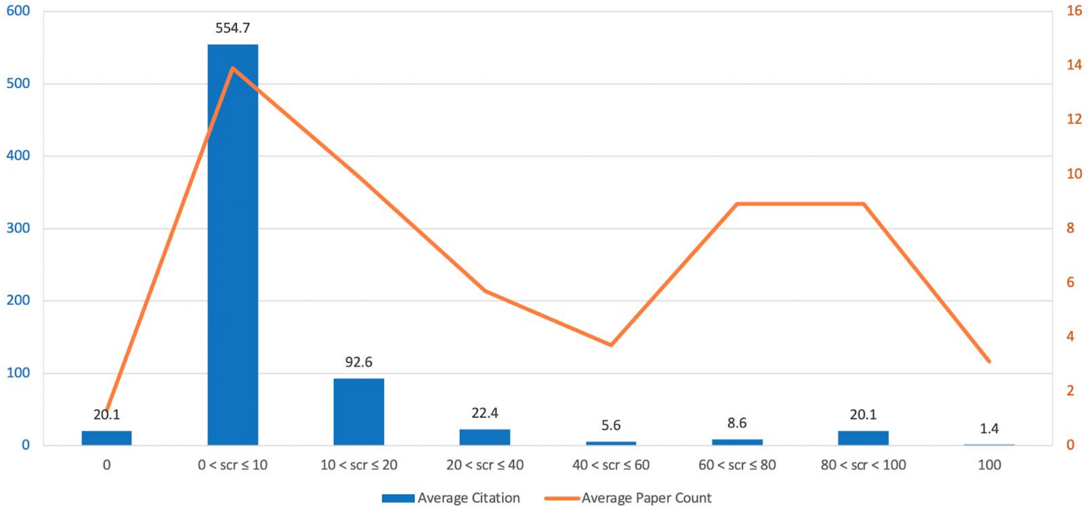


Figure 3.2.2 Average Citation and Paper Count versus Self-Citation Rate

It can be observed from the chart that authors with low SCR have more citations and papers published, except for the extreme cases, where for authors. For authors with less than 10% and 20% SCR, the number of citations and paper counts are obviously higher than others. This implies the negative co-relationship between SCR and citations/paper published. It is also worth noting that authors with SCR of 0% have the lowest paper published, and relatively low citation counts compared to authors with low SCR. This implies that the number of citations is positively related to the continuity of one's research field, where one's research work and reputation is built upon his/her previous works.

3.2.2 Self-Citation Analysis Based on Topic Detection with LDA

A further analysis is done to find out the differences in research topics of authors with high and low SCR, using the topic detection model with LDA in section 2.2.1.

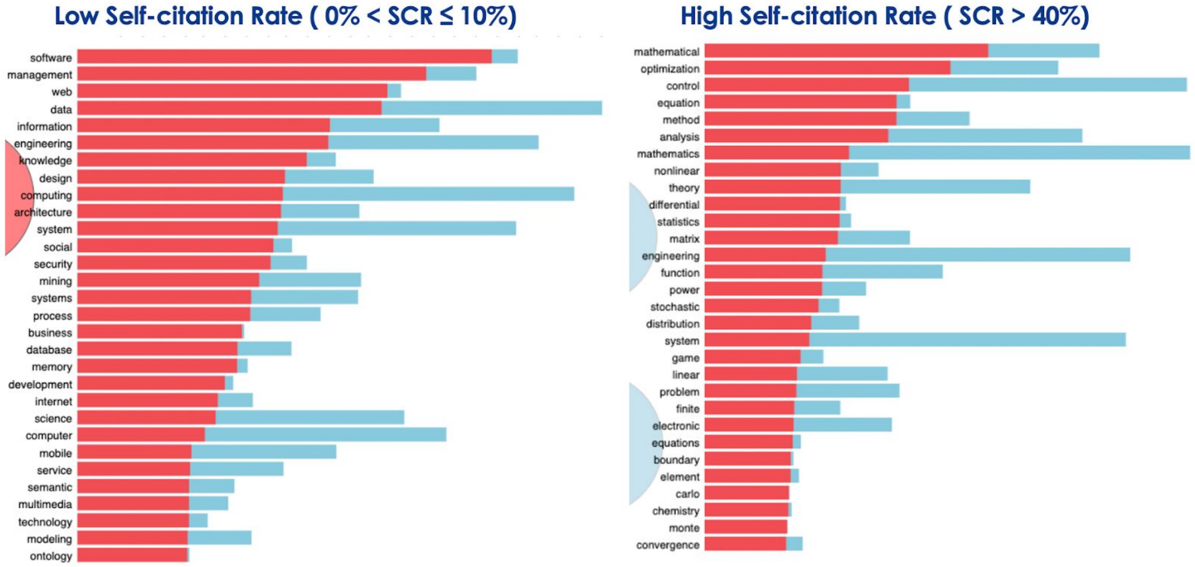
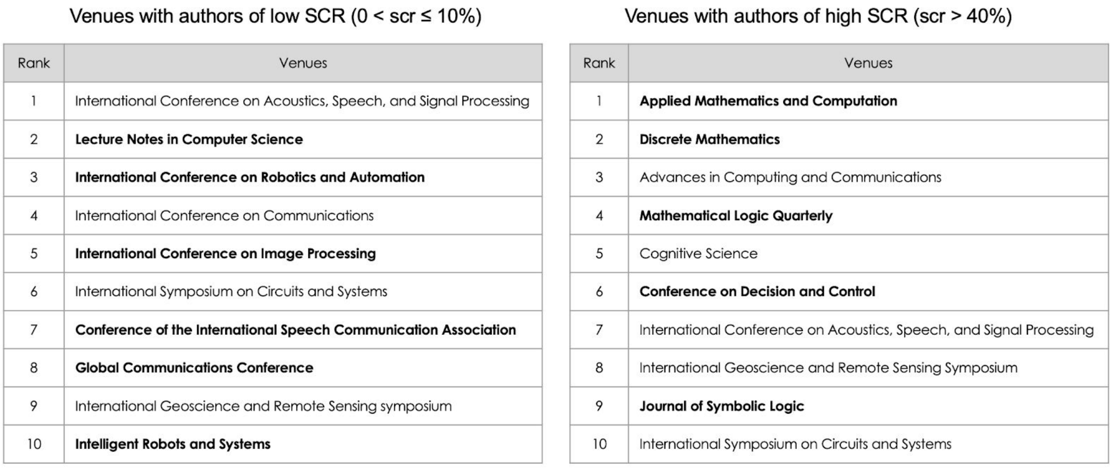


Figure 3.2.3: Results of topic detection with most frequent keywords of largest cluster

For authors with low SCR of less than 10%, their largest topic cluster consists of keywords which are more modern, popular, and applicable, such as web, software, information, data etc. On the other hand, authors with a high SCR have the largest cluster consisting of keywords which are more traditional and theoretical focus, such as mathematics, statistics, game theory etc.

3.2.3 Self-Citation Analysis Based on Venues

To further strengthen and validate the observation from the result of topic detection, another analysis was performed based on the venues where authors' paper being published, and compare the venue ranking on both cases.



Venues with authors of low SCR (0 < scr ≤ 10%)

Rank	Venues
1	International Conference on Acoustics, Speech, and Signal Processing
2	**Lecture Notes in Computer Science**
3	**International Conference on Robotics and Automation**
4	International Conference on Communications
5	**International Conference on Image Processing**
6	International Symposium on Circuits and Systems
7	**Conference of the International Speech Communication Association**
8	**Global Communications Conference**
9	International Geoscience and Remote Sensing symposium
10	**Intelligent Robots and Systems**

Venues with authors of high SCR (scr > 40%)

Rank	Venues
1	**Applied Mathematics and Computation**
2	**Discrete Mathematics**
3	Advances in Computing and Communications
4	**Mathematical Logic Quarterly**
5	Cognitive Science
6	**Conference on Decision and Control**
7	International Conference on Acoustics, Speech, and Signal Processing
8	International Geoscience and Remote Sensing Symposium
9	**Journal of Symbolic Logic**
10	International Symposium on Circuits and Systems

Figure 3.2.4: Top venues with most papers from authors with low/high SCR

The figure above consists of the top10 ranking of venues with most papers from authors with low and high SCR. For venue ranking on the right consisting of most papers from authors with low SCR, it can be observed that these venues are mostly focusing on more applicable and recent research trends such as CV and NLP. In contrast on the right, venues with most papers from authors with high SCR, venues traditional and theoretical research areas such as mathematics and logics.

3.2.4 Self-Citation Analysis Summary

Self-citation rate is highly correlated to popularity and area of one's research topics, amount of research work done, and continuity of previous research, such that

- Research work done (citations and papers count) is negatively correlated to self-citation rate.
- Popularity of research topics is negatively correlated to self-citation rate.
- Authors with extremely low (very close to 0%) self-citation rate, are very likely to have little papers published and citations.

Based on the results of **topic-detection** and **venue** comparison,

- Authors with high SCR tend to be involved in more traditional and theoretical research topics such as Mathematics, Statistics, Logics etc.
- Authors with low SCR tend to work with more recent, popular and application-focus research topics such as CV, NLP, Network, AI etc.

Further details can be referred to the self-citation clusters, where visualization of topics for different self-citation rate groups is provided.

3.3 Community Detection

Modularity is a metric to measure the strength of division of a network into communities. The network with dense intra-community links and sparse inter-community links will yield higher modularity score (Clauset et al., 2004). The score is ranging between 1 to -1. Value close to 1 indicates strong community structure, while value close to -1 shows weak community structure. When the score is zero, means the community division is not better than random. Modularity Equation is as follows:

$$Q = \sum_{c=1}^{n} \left[\frac{L_c}{m} - \left(\frac{k_c}{2m}\right)^2 \right]$$

The sum iterates over all communities c, m is the number of edges, L_C is the number of intra-community links for community c, k_C is sum of degrees of the nodes in community c.

In this project, modularity score is used to evaluate the performance of community detection algorithms. For each algorithm, every author will be assigned to their respective community label. Then, the modularity score will be calculated to identify the best algorithm.

Method	Time Complexity	Number of Clusters	Modularity Score
Label Propagation Algorithm (LPA)	$O(N)$	1045495	0.55
Connected Components (CC)	$O(N+E) \approx O(N)$	535420	0.20
Louvain Algorithm	$O(NlogN)$	536163	**0.85**

Table 3.3.1 Community Detection Results

Based on Table 3.3.1, LPA outputs the highest number of clusters, while the number of clusters for CC and Louvain is almost ½ of LPA. Both LPA and CC have near linear-time complexity O(N), therefore the computation time is fast (5min with Spark GraphFrames). In terms of modularity score, Louvain Algorithm has the highest modularity score, while Connected Components has the lowest modularity score. We will further analyse the reason behind these modularity scores by looking at the cluster size distribution of each algorithm.

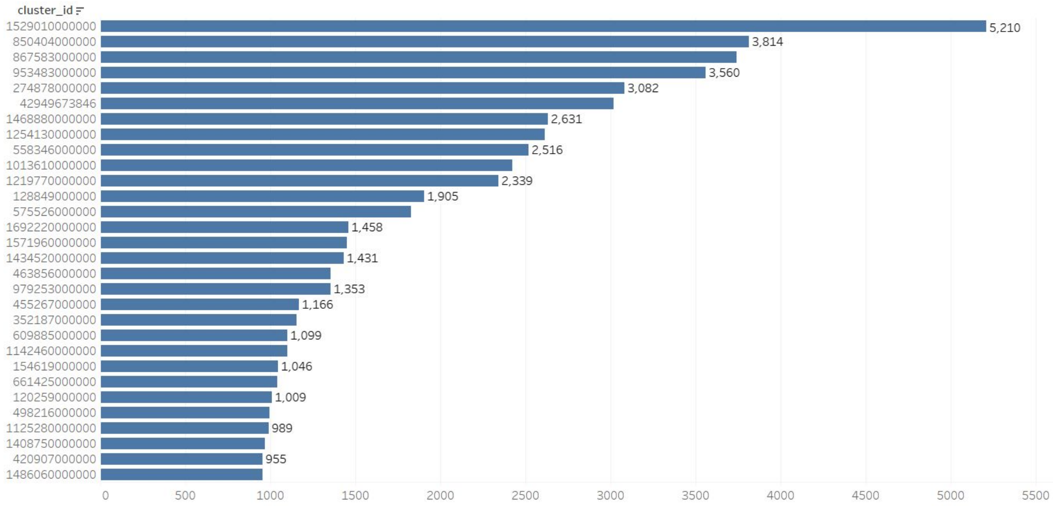


Figure 3.3.2 Top 30 Cluster Size - Label Propagation Algorithm (LPA)

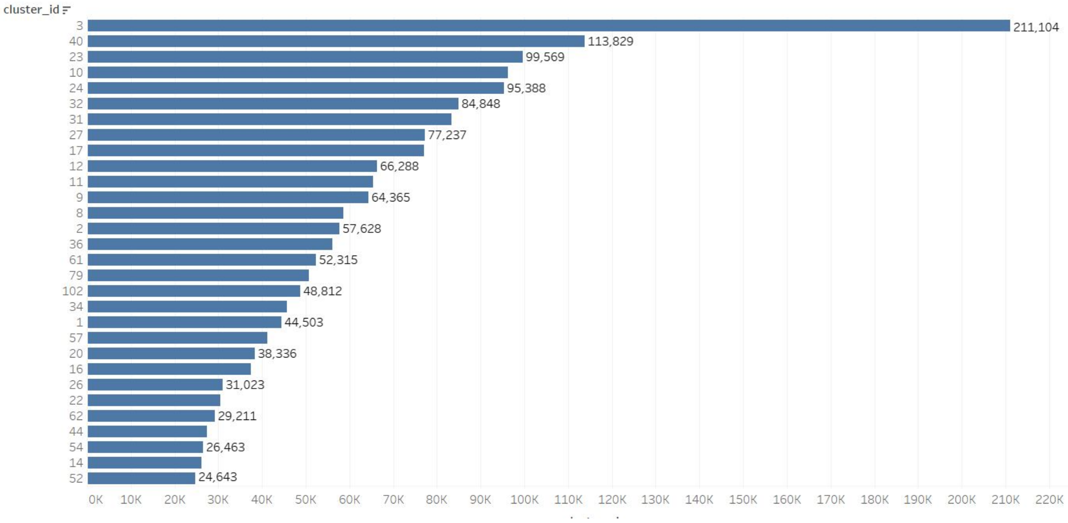


Figure 3.3.3 Top 30 Cluster Size - Louvain Algorithm

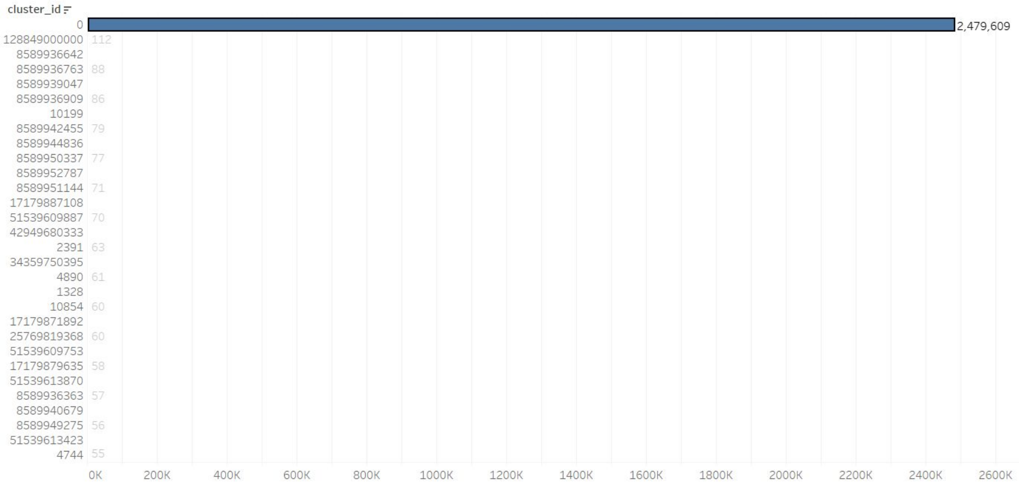


Figure 3.3.4 Top 30 Cluster Size - Connected Components (CC)

The figures above show the top 30 cluster size for all three algorithms. From here we can see that the Connected Components doesn't really work well with our network as the majority of the authors have at least one collaboration with other authors. This is because CC cluster formation requires only a link to exist between pairs of nodes, so it tends to form an extremely large cluster that fills most of the network. Eg. TopClusterSize=2,479,609. The rest of the nodes are divided into groups of small components disconnected from the large cluster. Hence, the output of CC is lack of modularity and thus resulting in lower modularity score (0.20).

On the other hand, the cluster size distribution of LPA and Louvain Algorithm are quite balanced across multiple clusters and this is why they have higher modularity scores, 0.55 and 0.85 respectively. Louvain outperformed LPA due to the core of the algorithm is to maximize the modularity for each community as the algorithm progresses. Since Louvain has the best modularity score, the rest of the analysis will be done using its clustering result.

With the clustering result, we can find out which communities have high co-authorship, where the authors tend to collaborate with each other very often within the same community. This task can be achieved by calculating the average of the intra-community links (collaboration count) for each cluster.

Cluster ID	Cluster Size	Mean Collaboration Count	Mean Triangle Count
5	3664	3.7	5872.47
2622	192	2.5	316.89
12	66288	2.01	37.61
40	113829	1.96	26.89
27	77237	1.95	21.51

Table 3.3.5 Top 5 Highest Collaboration Count Clusters

Table 3.3.5 shows the top 5 highest collaboration count clusters in the network. The clusters that have high collaboration counts, means they have more high-weighted links. We also realized that clusters that have more high-weighted links tend to have higher triangle counts too. Triangle Count is the number of triangles passing through each node. A triangle is a set of three nodes, where each node has a link to all other nodes. Therefore, a higher triangle count implies that the community has higher cohesion, where every author has a link to all other authors within the same community. Hence, we can conclude that the authors in Cluster 5, 2622, 12, 40 and 27 tend to collaborate with each other very often within the same community.

Other than that, we can also identify which communities are more authoritative (trustable) by calculating the pagerank for each node, then take the average pagerank of each cluster.

Cluster ID	Cluster Size	Mean PageRank
5769	164	1.58
322	3464	1.46
3	211104	1.37
75	2060	1.34
1311	956	1.34

Table 3.3.6 Top 5 Highest Pagerank Clusters

Table 3.3.6 shows the top 5 highest pagerank clusters in the network. The pagerank of an author is calculated recursively based on the number of incoming links (collaborations from the other authors). A community which has higher pagerank indicates that its authors have collaborations with many other high pagerank authors in the network, and therefore the community is said to be more trustable. Hence, we can conclude that the authors in Cluster 5769, 322, 3, 75 and 1311 are more trustable compared to the rest of the authors in the network.

Next, we intend to find out what are the popular topics among large and small communities. This task can be achieved by extracting the title and key terms from the papers published by each author, then group by clusterID to get the frequent words in each cluster. To show more meaningful results, we build a WordCloud by combining the frequent words from top 100 large and small clusters.

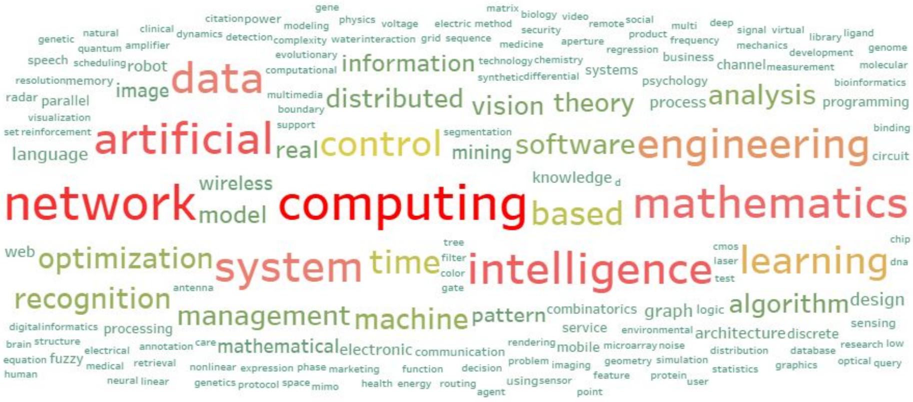


Figure 3.3.7 WordCloud of Frequent Words in Large Clusters

Based on Figure 3.3.7, it is very obvious that the top 5 frequent words in large clusters are computing, network, artificial, intelligence and data. Hence, we can conclude that Big Data and AI related papers are very popular among the large communities. These papers have more co-authorship in the academic network.

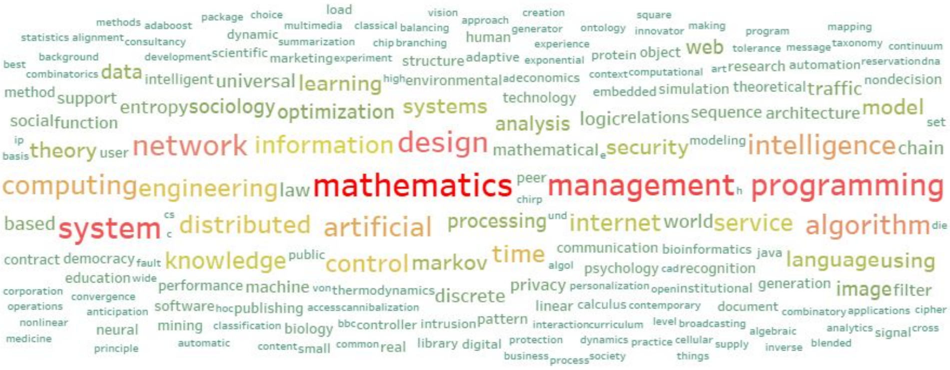


Figure 3.3.8 WordCloud of Frequent Words in Small Clusters

Top 5 frequent words in small clusters are mathematics, management, programming, system and design. Thus, we can deduce that the Mathematics, System Design, and Programming related papers have higher popularity among the small communities. These papers tend to have lesser co-authorship in the academic network.

3.4 Link Prediction

3.4.1 Similarity-based Approach

The link prediction results from the similarity-based approach are shown in Table 3.4.1. The top five best-performed metrics are Salton Cosine Similarity (0.706677), Jaccard Coefficient (0.706656), Resource Allocation (0.706651), Adamic Adar (0.706648) and Common Neighbors (0.706570). Salton Cosine Similarity and Common Neighbors are preferred as they have less time complexity but higher performance. This has set a baseline AUC of 0.70667 for the link prediction task in this project.

Method	**Time Complexity**	**Train AUC**	**Test AUC**
Common Neighbors	$O(N^2)$	0.680020	0.706570
Total Neighbors	$O(N^2)$	0.641919	0.639651
Preferential Attachment	$O(2N)$	0.648356	0.645873
Jaccard Coefficient	$O(2N^2)$	0.680073	0.706656
Adamic Adar	$O(2N^2)$	0.680063	0.706648
Resource Allocation	$O(2N^2)$	0.680065	0.706651
Leicht Holme Nerman	$O(N^2)$	0.500217	0.500046

Sorensen Index	$O(N^2)$	0.500000	0.500000
Salton Cosine Similarity	**$O(N^2)$**	**0.680080**	**0.706677**
Hub Promoted	$O(N^2)$	0.511942	0.506438
Hub Depressed	$O(N^2)$	0.500386	0.500109

Table 3.4.1 Link Prediction Results - Similarity-based Approach

3.4.2 Learning-based Approach

The performance of the machine learning models trained by using neighbor-based features are shown in Table 3.4.2. Among all the models, Gradient Boosted Trees have produced the best result of 0.771244 on the test dataset but also taken the longest training time of 406 seconds. Although XGBosst has a slightly lower AUC value but the time taken is about half of Gradient Boosted Trees' training time. From this perspective, XGBoost is more efficient. It has improved the performance significantly from the baseline AUC of 0.706677 to 0.768801.

Method	Train AUC	Validation AUC	Test AUC	Runtime (Seconds)
Logistic Regression	0.754961	0.752269	0.767290	159
Support Vector Machine	0.754516	0.751773	0.765609	89
Random Forest	0.752896	0.750323	0.767127	759
Gradient Boosted Trees	0.755912	0.753134	0.768370	441
XGBoost	**0.756476**	**0.753553**	**0.768801**	**173**

Table 3.4.2 Link Prediction Results - Learning-based Approach with Neighbor-based Features

The performance of the models trained by using both neighbor and community-based features are shown in Table 3.4.3. XGBoost has produced the best result with a runtime of 218 seconds. It has improved the performance slightly from the previous AUC of 0.768801 to 0.779786. The feature importance is illustrated in Figure 3.4.4. Jaccard and Salton Cosine Similarity are the most important similarity features for link prediction, which is aligned with the results from similarity-based approach, while triangle count is the most important community-based feature for link prediction.

Method	Train AUC	Validation AUC	Test AUC	Runtime (Seconds)
Logistic Regression	0.763069	0.765381	0.777019	57
Support Vector Machine	0.758757	0.760956	0.774376	130
Random Forest	0.762601	0.764675	0.776551	407
Gradient Boosted Trees	0.766560	0.767408	0.779431	412
XGBoost	**0.769488**	**0.767833**	**0.779786**	**218**

Table 3.4.3 Link Prediction Results - Learning-based Approach with Neighbor and Community-based Features

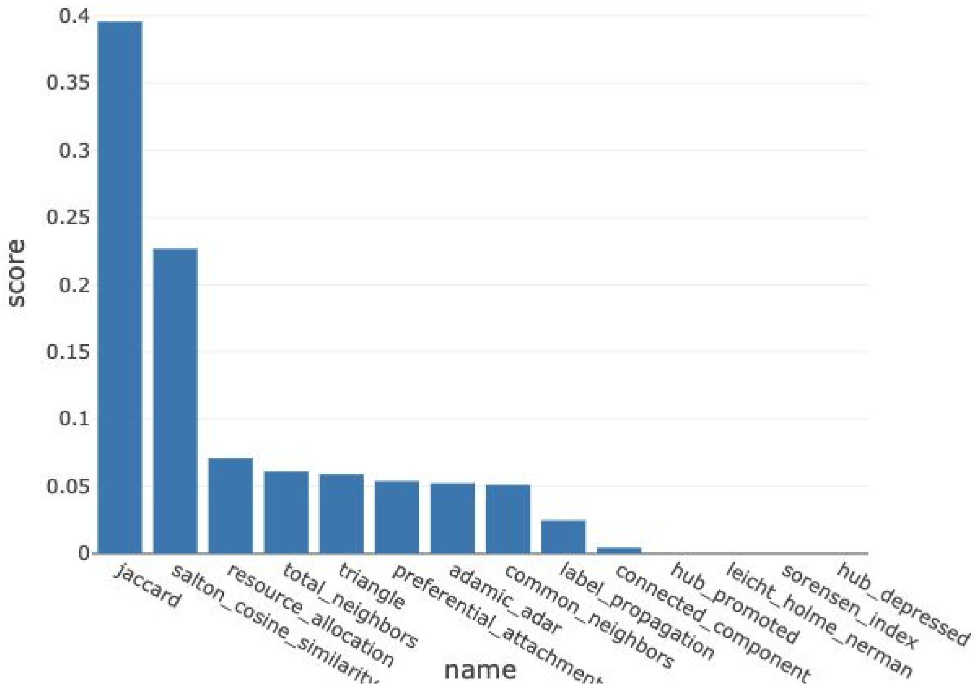


Figure 3.4.4 Feature Importance - Learning-based Approach with Neighbor and Community-based Features

Next, the node embeddings generated from the Node2Vec algorithm are used as additional feature representations to train the models. Generally, there are two sets of parameters in Node2Vec algorithms. One is for random walks and the other one is for Word2Vec. For Word2Vec, the following parameters are selected and fixed in all experiments: minCount: 10, maxIter: 50, windowSize: 5, vectorSize: 64. The parameters for random walks are fine-tuned which include the number of sample walks starting from each node (num_walks), the length of each random walk path (walk_length), p (return_param) and q (inout_param) which prioritizes the BFS and DFS procedure respectively. Table 3.4.5 has shown the prediction results when the parameters are set as num_walks: all, walk_length: 1, return_param: 1, inout_param: 1. It has improved the performance slightly from the previous AUC of 0.779786 to 0.801189. Once again, XGBoost is the most efficient algorithm. Figure 3.4.6 has ranked the feature importance after including the node embeddings. The node embeddings have emerged as the important factors but are not that significant. The top 6 important features are still from neighbor and community-based features, which indicates that the parameters selected for random walks are not that effective.

Method	Train AUC	Validation AUC	Test AUC	Runtime (Seconds)
Logistic Regression	0.789407	0.790284	0.796665	271
Support Vector Machine	0.764115	0.764254	0.764686	2,274
Random Forest	0.789527	0.788928	0.793454	1,919
Gradient Boosted Trees	0.813819	0.804585	0.799590	4,668
XGBoost	**0.836042**	**0.812962**	**0.801189**	**1,361**

Table 3.4.5 Link Prediction Results - Learning-based Approach with Neighbor and Community-based Features and Node Embeddings (num_walks: all, walk_length: 1, return_param: 1, inout_param: 1)

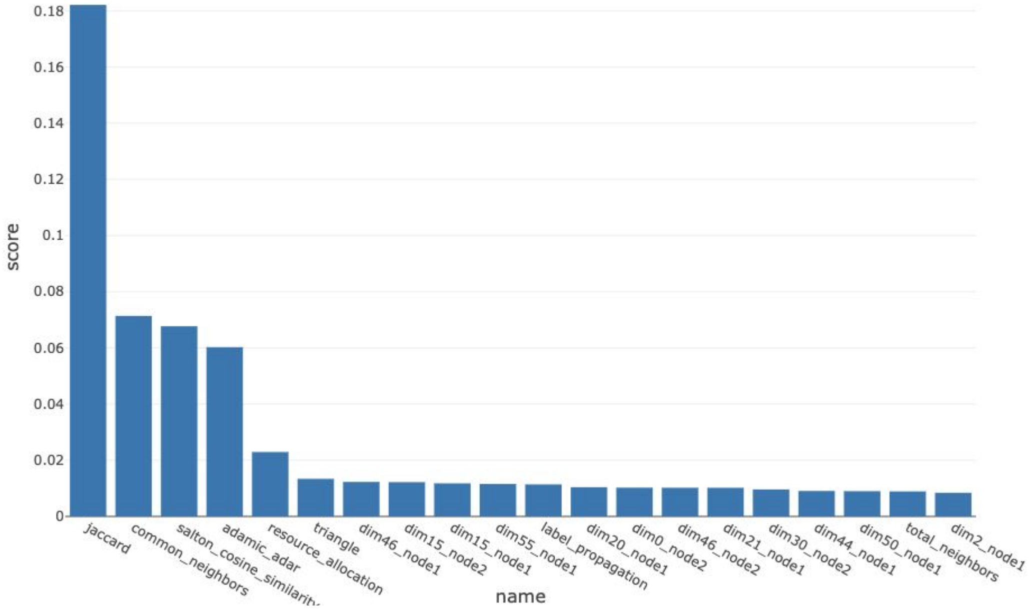


Figure 3.4.6 Feature Importance - Learning-based Approach with Neighbor and Community-based Features and Node Embeddings (num_walks: all, walk_length: 1, return_param: 1, inout_param: 1)

The next step is to increase the length of random walks and examine its effect on model results. Table 3.4.7 shows that the performance has been boosted from 0.801189 to 0.865415 by XGBoost with a runtime of 2,324 seconds. In terms of feature importance, Salton Cosine Similarity and Jaccard remain the most important features but more node embeddings have become important factors (Figure 3.4.8).

Method	Train AUC	Validation AUC	Test AUC	Runtime (Seconds)
Logistic Regression	0.770285	0.770436	0.775702	279
Support Vector Machine	0.768641	0.768552	0.771185	948
Random Forest	0.775752	0.773810	0.777152	1,505
Gradient Boosted Trees	0.855001	0.849737	0.849324	3,609
XGBoost	**0.881273**	**0.872523**	**0.865415**	**2,324**

Table 3.4.7 Link Prediction Results - Learning-based Approach with Neighbor and Community-based Features and Node Embeddings (num_walks: 20, walk_length: 10, return_param: 1, inout_param: 1)

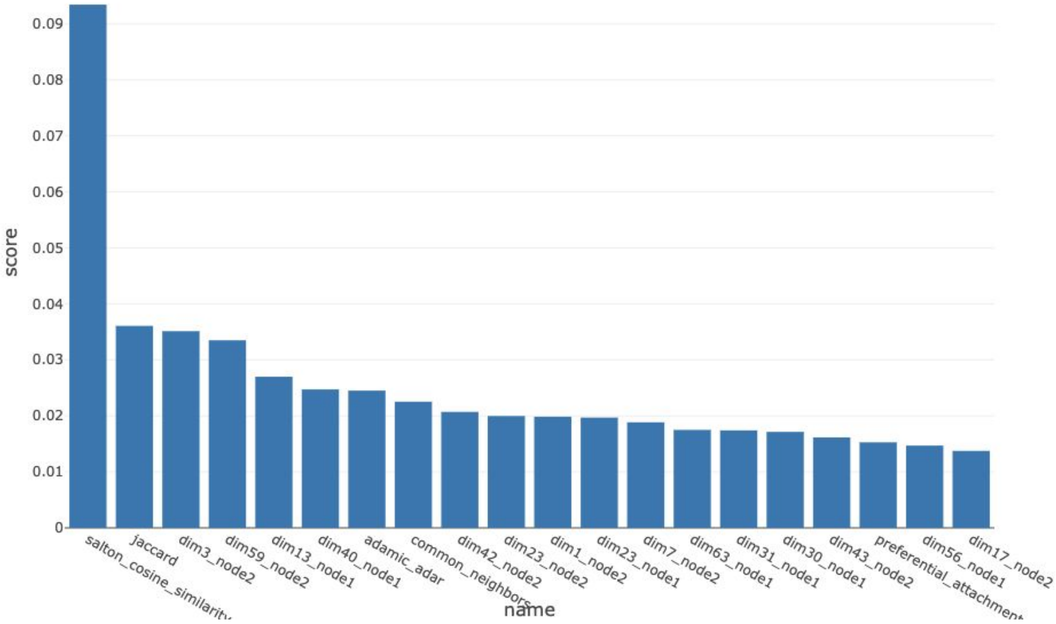


Figure 3.4.8 Feature Importance - Learning-based Approach with Neighbor and Community-based Features and Node Embeddings (num_walks: 20, walk_length: 10, return_param: 1, inout_param: 1)

Increasing the length of walks and sample size can increase the model performance and stability, which is further proved by the next experiment where the following parameters are chosen: num_walks: 20, walk_length: [10, 15] return_param: 1, inout_param: 1. Due to the time constraint, other parameters for Node2Vec are not further experimented and the model has been achieved has an AUC of 0.871546 (Table 3.4.9).

Method	Train AUC	Validation AUC	Test AUC	Runtime (Seconds)
Logistic Regression	0.771268	0.770538	0.776241	304
Support Vector Machine	0.769810	0.768339	0.771357	763
Random Forest	0.778418	0.776673	0.784980	1,771
Gradient Boosted Trees	0.859426	0.854431	0.857487	2,251
XGBoost	**0.886222**	**0.877596**	**0.871546**	**2,526**

Table 3.4.9 Link Prediction Results - Learning-based Approach with Neighbor and Community-based Features and Node Embeddings (num_walks: 20, walk_length: [10, 15] return_param: 1, inout_param: 1)

Beside features derived from graph, node properties are also important for link prediction. As the last attempt to further improve the model performance, one of the available node properties, research interests, are included in the link prediction. The assumption is that authors who have similar research interests tend to collaborate. The key terms of all the papers published before authors collaborate are extracted (i.e. papers published before 2012 for training data and before 2015 for test data). The TF-IDF matrix is constructed to derive the cosine similarity between the two authors based on the key terms stated in their published papers. It turns out that it is a very important factor in co-authorship prediction and the final model result has been boosted from 0.871546 to 0.930396 (Table 3.5.10).

Method	Train AUC	Validation AUC	Test AUC	Runtime (Seconds)
Logistic Regression	0.912060	0.911161	0.920562	545
Support Vector Machine	0.909211	0.908050	0.917308	725
Random Forest	0.888552	0.887469	0.896284	1,430
Gradient Boosted Trees	0.923759	0.921933	0.929657	2,188
XGBoost	**0.924288**	**0.923079**	**0.930396**	**1,191**

Table 3.4.10 Link Prediction Results - Learning-based Approach with Neighbor and Community-based Features, Node Embeddings (num_walks: 20, walk_length: [10, 15] return_param: 1, inout_param: 1) and Cosine Similarity of Research Interests

This section has implemented the link prediction on Spark and improved the co-authorship prediction performance from 0.706677 in the similarity-based approach to 0.930396 in the learning-based approach by including node embeddings and node properties. It is observed that community-based features do not improve the model performance significantly. The reason could be that neighbor-based similarity metrics have already carried community information. For instance, authors who have many common neighbors have a high chance of belonging to the same community. Node embeddings can improve the model performance significantly. The length of walk and number of samples are two important parameters to fine-tune Node2Vec. It is also noticed that node embeddings alone do not produce promising results but can boost the model performance after combining with neighbor and community-based features. Among all the classifiers, boosting algorithms, particularly XGBoost, are more effective in running large dimension classification tasks. To the best of our knowledge, no prior study has been using the entire AMiner citation network to perform large-scale co-authorship prediction. In terms of performance benchmarking, the latest models which are trained by the DBLP dataset, which is a subset of AMiner dataset, have achieved an AUC of 0.847 (Fu, Hou, et al., 2019) and 0.901 (Fu, Yuan, et al., 2019). If given time, the current experiments could be run on the DBLP dataset and compare the results.

3.4.3 Graph Neural Networks

Experiment Setting for Deep Learning

We compare different models including Linear Neural Network, basic GCN, GraphSage and Scalable GCN on our dataset. I have 11 types of features including common_neighbors, total_neighbors, preferential_attachment, jaccard, adamic_adar, resource_allocation, leicht_holme_nerman, sorensen_index, salton_cosine_similarity, hub_promoted, hub_depressed. They all generated from the above task or original from the dataset. Our dataset for this part contains 92497 distinct authors, 955572 links. I use an SGD optimizer with 0.01 learning rate and 0.5 momentum. The training and testing is realized on microsoft databricks and on the python, pyspark, pytorch environment.

Result and Analysis

layers	model	Test-F1	RunTime/batch
2	GraphSage	0.9415	0.120
2	ScalableGCN	0.9384	0.026
3	GraphSage	0.9482	1.119
3	ScalableGCN	0.9433	0.035

Table 3.4.11 Compare GraphSage and Scalable GCN

The experiment on the open dataset Reddi shows that the performance of GraphSage and ScalableGCN is quite similar while the scalable GCN consumes less time than that of GraphSage. It fits what ScalableGCN wants to do and shows the power of the model.

model	Recall	Precision	F1	Accuracy
Linear	99.65	50.08	66.66	50.14
basic GCN	57.84	88.65	70.01	58.31

Table 3.4.12 The result of Linear Neural Network and Graph Convolutional Network on our dataset

The experiment shows the deep learning model still needs to be further tuned. The Linear neural network result is not very good. But as the time and resource limits, we may do further research on the result of the model. One possible problem is the unbalanced dataset: most nodes are negative while few nodes are positive. It results in the machine learning model tends to predict all nodes with negative. The GCN model shows a better performance. It shows that the information of the graph does work for prediction.

model	Runtime
Without distributed	682s
With distributed Horovod	435s

Table 3.4.13 The performance of distributed training tools Horovod on MNIST

Our experiment uses 2 cores when distributed training and tests 100 batches. The results show that the Horovod tool runs quicker as the models can be trained on multiple cores. That is because each machine in the cluster has its own gradient (local gradient), and it needs to use the AllReduce mentioned above to complete the global gradient merging. Optimizer is the key API for model training, which can obtain the gradient of each optimizer and use it to update the weight. Another important part is how to update gradients synchronously. After the gradients of all ranks are calculated, the global gradient accumulation is performed uniformly. This involves message communication in the cluster. For this reason, Horovod has done two aspects of work. First, rank0 is used as the coordinator in Horovod, and rank0 coordinates the progress of all Ranks. Second, in order not to block normal calculations, a background communication thread is created in Horovod, which is specifically used for message synchronization between ranks and AllReduce operations.

3.4.4 Distributed Training

As we have sent email to the professor, it is not allowed to use GPU in the microsoft databricks. We test the pytorch package "DataParallel" and "DistributedDataParallel" on our own GPU. As the resources are limited, we only run some tests to compare the different techniques. For the Horovod, the microsoft databricks support the use of harovod for distributed training of deep learning models. We test on the cluster and multiple CPUs for this tool.

Method	Main	Auxiliary	Train time
DataParallel	3867MB	1982MB	3537s
DistributedDataParallel	1536MB	1536MB	517s

Table 3.5.1 Distribute Training Results - on 2 GPUs

Batchsize	Main	Auxiliary
64	3611MB	1833MB
128	3712MB	1846MB
256	3867MB	1982MB

Table 3.5.2 Distribute Training Results - on 2 GPUs

My experiment shows that the DistributedDataParallel shows less training time and more balance for the GPU memory cost. When the batch size increases, the memory cost will also increase. Unlike DataParallel's single-process control of multiple GPUs, with the help of distributed, we only need to write a piece of code, and torch will automatically allocate it to n processes and each is running on each GPUs.

Besides, in Dataparallel, data is directly divided into multiple GPUs, and data transmission will greatly affect efficiency. In contrast, using a sampler in DistributedDataParallel can divide a part of the data set for each process and avoid data duplication between different processes.

4. Problems Encountered and Lesson Learnt

Problems encountered in influencer identification & self-citation analysis :

- The dirty data in author-level from original data is hidden and hardly observed, for example, there are 3k papers with duplicate author name and id, which might be caused by duplicating name in the same research lab being identified as the same author.
- The computation for running complete PageRank algorithms for a full set of co-citation networks is extremely long (12 hours and failed). More consideration such as pre-estimation and planning on such experiments should be considered, such as calculating an estimated converging point based on analysis needs and limited budgets.

Problems encountered in community detection:

- Modularity score is widely used to measure the clustering quality. However, it is not available in the GraphFrames library. The only available API in python is *"networkx.algorithms.community.quality.modularity"*. Due to sequential coding, it is very slow and it took more than 6 hours to calculate the modularity score for all three algorithms. Therefore, in order to speed up the computations, we decided to study the networkx codes and rewrite it in spark to utilize the parallelism. The new spark program is now able to compute the modularity score for all three algorithms in around 4 minutes. Moreover, we also did a simple sanity check by comparing the outputs from networkx API and new spark program using a small network.
- Louvain algorithm is a popular community detection algorithm. Unfortunately, it is also not available in the GraphFrames library. Therefore, we have to implement this algorithm using the networkx library API. The computation time is long but still acceptable. It took around 2 hours to complete the calculations.

Problems encountered in link prediction:

- Not all graph algorithms have libraries on Spark and we have to implement the entire algorithm. For example, all the similarity metrics are implemented in Scala.

- Small clusters are selected for Databricks due to resource constraint and long runtime has been encountered particularly for Node2Vec. For example, it has taken up to 1.5 hours to generate the node embeddings. While autocalsing is economic but it is noticed that the resizing is taking a long time to acquire new nodes.
- More parameters tuning (e.g. dimension of node embeddings) should be performed for Node2Vec but resources and time are constraints. Edge-based features such as shortest paths can be used as features to perform link prediction.
- More research needs to be done about graph neural networks as the performance of the model seems not good in our experiment but resources and time are limited.

5. Personal Contribution

Name	Contribution
Cai Yufan	Link prediction using graph neural network and distributed training
Chong Sai King	Community detection algorithms survey, coding, experiment and result analysis.
Du Zidong	The similarity-based and feature-based learning approaches for link prediction and Power BI dashboard development.
Huang Xuankun	Self-citation detection and analysis, author-level data summary, and influencers identification.
Khoo Ju Seng Bryant	Text mining and LDA for topic modelling, visualisations of topics using pyLDAvis.

6. Summary

With the increasing demands in SNA and the larger size and complexity of big graph data, the mining and processing of such data has seen its limitation on in-memory computation. There are various big data processing methods thus introduced to resolve SNA on a large scale basis. Academic Social Network Analysis (ANSA), as a variant of SNA in context of scholarly data, promotes academic engagement and influences intellectual performance. This project has provided a comprehensive analysis of the academic social networks on popular SNA tasks using Spark. It is a challenging task with implementation of various algorithms on Spark. Based on our research of the past studies in ASNA, this is the first attempt to implement a comprehensive analysis of academic social networks on a distributed cluster-computing framework. Lastly, this project has developed a SNA framework (i.e. from data extraction, transformation, analysis to visualization) that could be used to perform similar tasks with other datasets.

7. References

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. In *Journal of Statistical Mechanics: Theory and Experiment* (Vol. 2008, Issue 10, p. P10008). https://doi.org/10.1088/1742-5468/2008/10/p10008

Clauset, A., Newman, M. E. J., & Moore, C. (2004). Finding community structure in very large networks. In *Physical Review E* (Vol. 70, Issue 6). https://doi.org/10.1103/physreve.70.066111

Fu, G., Hou, C., & Yao, X. (2019). Learning Topological Representation for Networks via Hierarchical Sampling. In *2019 International Joint Conference on Neural Networks (IJCNN)*. https://doi.org/10.1109/ijcnn.2019.8851893

Fugue: Unifying Spark and Non-Spark Ecosystems for Big Data Analytics. (n.d.). Retrieved November 26, 2020, from https://databricks.com/session_na20/fugue-unifying-spark-and-non-spark-ecosystems-for-big-data-analytics

Fu, G., Yuan, B., Duan, Q., & Yao, X. (2019). Representation Learning for Heterogeneous Information Networks via Embedding Events. In *Neural Information Processing* (pp. 327–339). https://doi.org/10.1007/978-3-030-36708-4_27

Grover, A., & Leskovec, J. (2016). node2vec: Scalable Feature Learning for Networks. *KDD: Proceedings / International Conference on Knowledge Discovery & Data Mining. International Conference on Knowledge Discovery & Data Mining*, *2016*, 855–864.

Levorato, V., & Petermann, C. (2011). Detection of communities in directed networks based on strongly p-connected components. In *2011 International Conference on Computational Aspects of Social Networks (CASoN)*. https://doi.org/10.1109/cason.2011.6085946

Lichtenwalter, R. N., Lussier, J. T., & Chawla, N. V. (2010). New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '10*. https://doi.org/10.1145/1835804.1835837

Needham, M., & Hodler, A. E. (n.d.). *Graph Algorithms*. O'Reilly Media, Inc.

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). *DeepWalk: Online Learning of Social Representations*. https://doi.org/10.1145/2623330.2623732

Raghavan, U. N., Albert, R., & Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physical Review. E, Statistical, Nonlinear, and Soft Matter Physics*, *76*(3 Pt 2), 036106.

Soric, I., Dinjar, D., Stajcer, M., & Orescanin, D. (2017). Efficient social network analysis in big data architectures. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. https://doi.org/10.23919/mipro.2017.7973640

Tang, J., Zhang, J., Yao, L., & Li, J. (2008). Extraction and mining of an academic social network. In *Proceeding of the 17th international conference on World Wide Web - WWW '08*. https://doi.org/10.1145/1367497.1367722

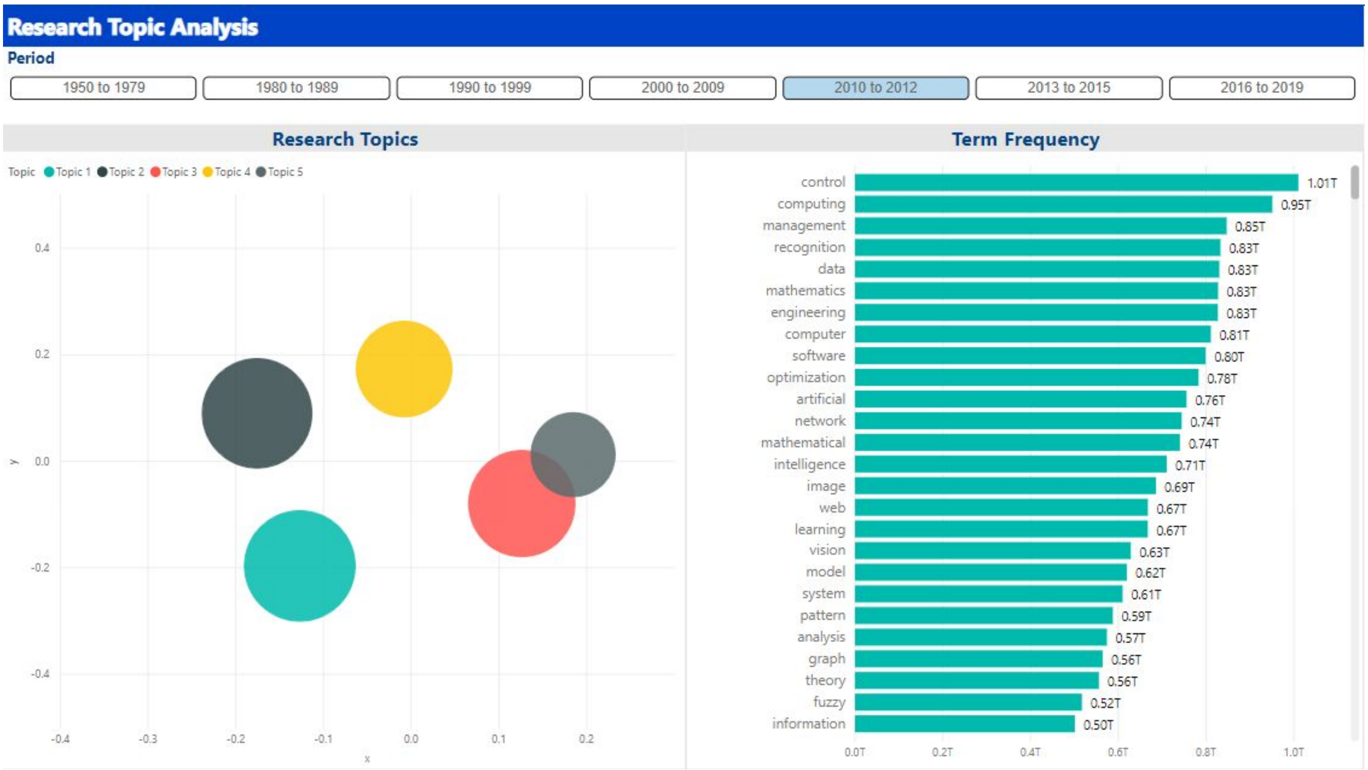
Wang, P., Xu, B., Wu, Y., & Zhou, X. (2015). Link prediction in social networks: the state-of-the-art. In *Science China Information Sciences* (Vol. 58, Issue 1, pp. 1–38). https://doi.org/10.1007/s11432-014-5237-y

8. Workloads of Each Group Member

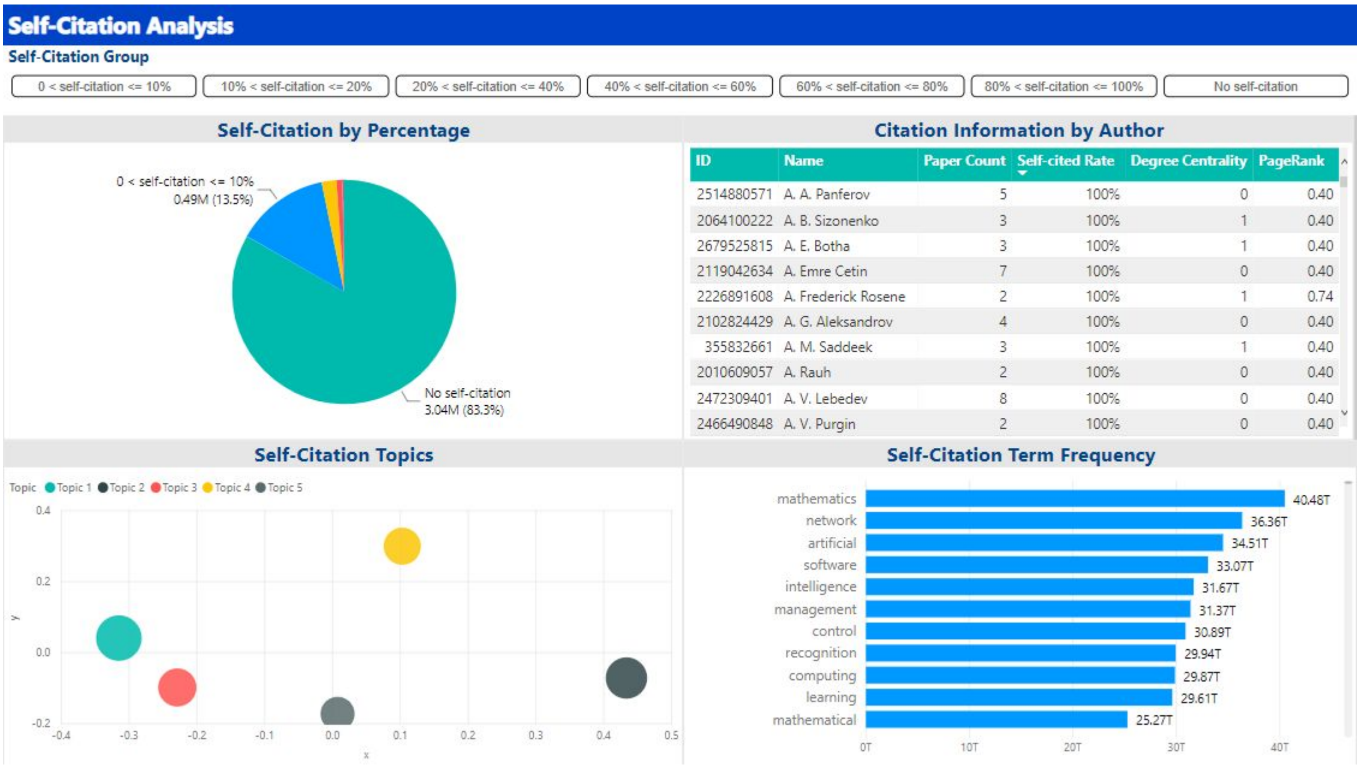
The workload is almost equally distributed among the team members.

9. Appendix - Power BI Dashboard

1. Research Topic Analysis

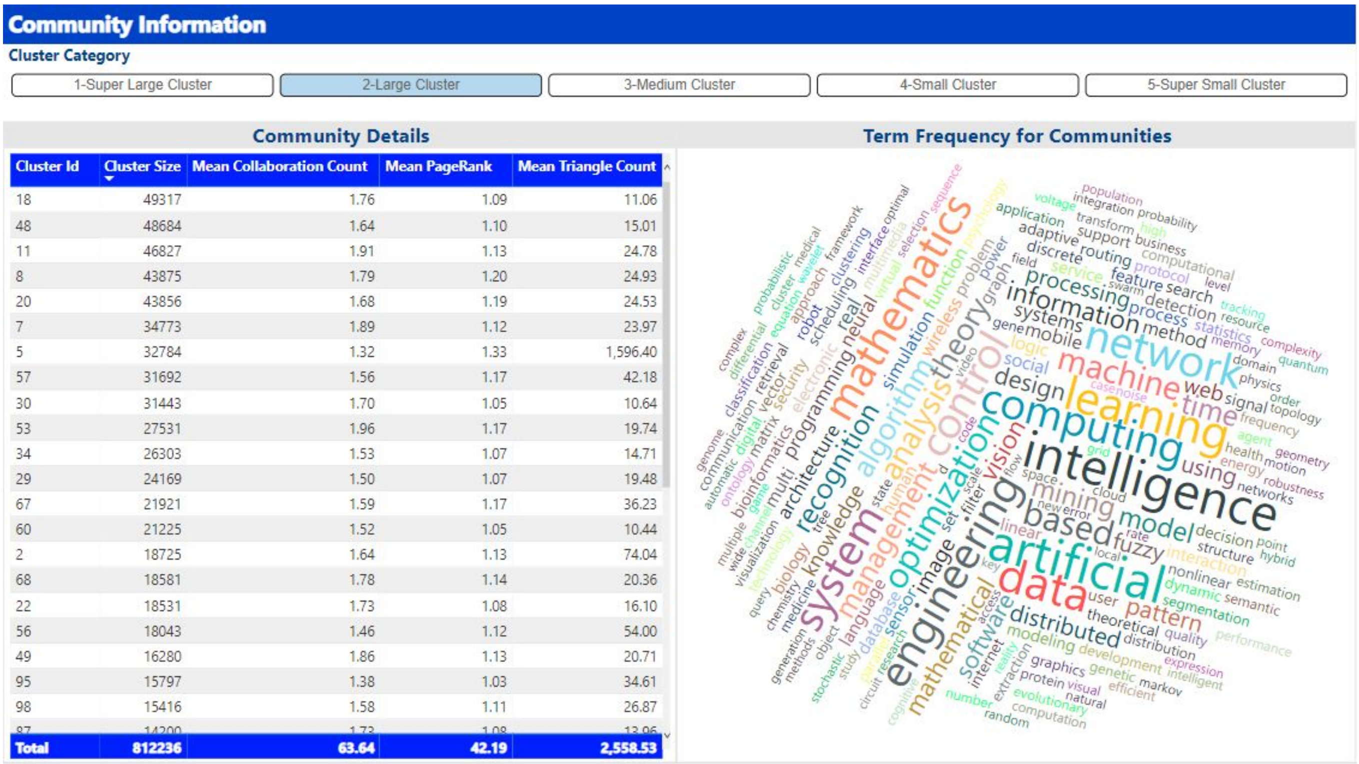


2. Self-Citation Analysis


Self-Citation Analysis
Self-Citation Group
0 < self-citation <= 10%
10% < self-citation <= 20%
20% < self-citation <= 40%
40% < self-citation <= 60%
60% < self-citation <= 80%
80% < self-citation <= 100%
No self-citation
Self-Citation by Percentage
0 < self-citation <= 10%
0.49M (13.5%)
No self-citation
3.04M (83.3%)
Citation Information by Author

ID	Name	Paper Count	Self-cited Rate	Degree Centrality	PageRank
2514880571	A. A. Panferov	5	100%	0	0.40
2064100222	A. B. Sizonenko	3	100%	1	0.40
2679525815	A. E. Botha	3	100%	1	0.40
2119042634	A. Emre Cetin	7	100%	0	0.40
2226891608	A. Frederick Rosene	2	100%	1	0.74
2102824429	A. G. Aleksandrov	4	100%	0	0.40
355832661	A. M. Saddeek	3	100%	1	0.40
2010609057	A. Rauh	2	100%	0	0.40
2472309401	A. V. Lebedev	8	100%	0	0.40
2466490848	A. V. Purgin	2	100%	0	0.40

3. Community Information



Community Information

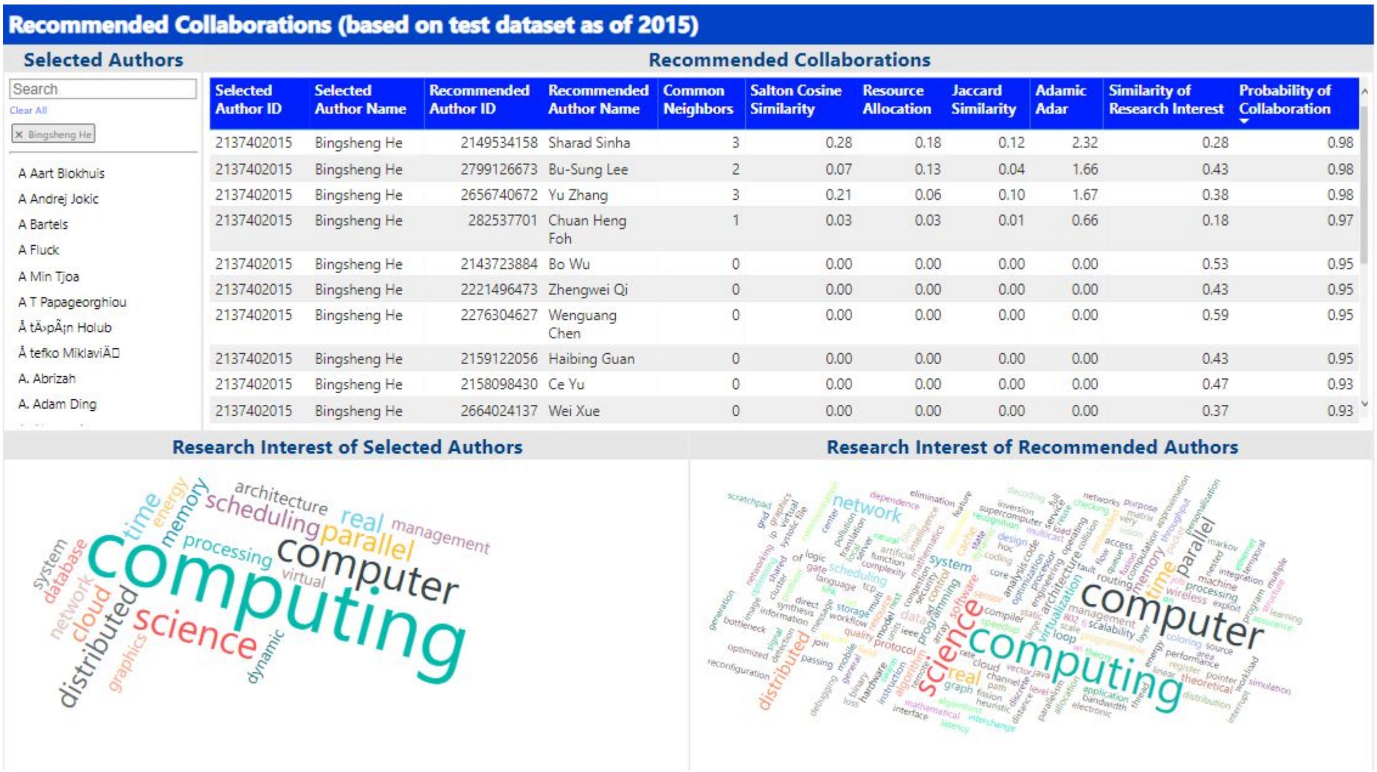
Cluster Category

Community Details

Cluster Id	Cluster Size	Mean Collaboration Count	Mean PageRank	Mean Triangle Count
18	49317	1.76	1.09	11.06
48	48684	1.64	1.10	15.01
11	46827	1.91	1.13	24.78
8	43875	1.79	1.20	24.93
20	43856	1.68	1.19	24.53
7	34773	1.89	1.12	23.97
5	32784	1.32	1.33	1,596.40
57	31692	1.56	1.17	42.18
30	31443	1.70	1.05	10.64
53	27531	1.96	1.17	19.74
34	26303	1.53	1.07	14.71
29	24169	1.50	1.07	19.48
67	21921	1.59	1.17	36.23
60	21225	1.52	1.05	10.44
2	18725	1.64	1.13	74.04
68	18581	1.78	1.14	20.36
22	18531	1.73	1.08	16.10
56	18043	1.46	1.12	54.00
49	16280	1.86	1.13	20.71
95	15797	1.38	1.03	34.61
98	15416	1.58	1.11	26.87
87	14200	1.73	1.08	13.96
Total	**812236**	**63.64**	**42.19**	**2,558.53**

Term Frequency for Communities

4. Recommended Collaborations



Recommended Collaborations (based on test dataset as of 2015)

Selected Authors

A Aart Blokhuis
A Andrej Jokic
A Bartels
A Fluck
A Min Tjoa
A T Papageorghiou
Å tÃ›pÃ¡n Holub
Å tefko MiklaviÄ
A. Abrizah
A. Adam Ding

Recommended Collaborations

Selected Author ID	Selected Author Name	Recommended Author ID	Recommended Author Name	Common Neighbors	Salton Cosine Similarity	Resource Allocation	Jaccard Similarity	Adamic Adar	Similarity of Research Interest	Probability of Collaboration
2137402015	Bingsheng He	2149534158	Sharad Sinha	3	0.28	0.18	0.12	2.32	0.28	0.98
2137402015	Bingsheng He	2799126673	Bu-Sung Lee	2	0.07	0.13	0.04	1.66	0.43	0.98
2137402015	Bingsheng He	2656740672	Yu Zhang	3	0.21	0.06	0.10	1.67	0.38	0.98
2137402015	Bingsheng He	282537701	Chuan Heng Foh	1	0.03	0.03	0.01	0.66	0.18	0.97
2137402015	Bingsheng He	2143723884	Bo Wu	0	0.00	0.00	0.00	0.00	0.53	0.95
2137402015	Bingsheng He	2221496473	Zhengwei Qi	0	0.00	0.00	0.00	0.00	0.43	0.95
2137402015	Bingsheng He	2276304627	Wenguang Chen	0	0.00	0.00	0.00	0.00	0.59	0.95
2137402015	Bingsheng He	2159122056	Haibing Guan	0	0.00	0.00	0.00	0.00	0.43	0.95
2137402015	Bingsheng He	2158098430	Ce Yu	0	0.00	0.00	0.00	0.00	0.47	0.93
2137402015	Bingsheng He	2664024137	Wei Xue	0	0.00	0.00	0.00	0.00	0.37	0.93

Research Interest of Selected Authors

Research Interest of Recommended Authors