Disambiguate, Model, Recommend & Visualise: A proof of concept toolkit to explore research collaboration

By

Liam Ephraims

Supervised by Dr. Timothy Churches

& Dr. Tanya Ward

Table of Contents

Word count (not including figures & tables): 10430 / 10000-word limit

1.0 Executive Summary

Over the past decade, research, medical and scientific bibliographic services such as PubMed, Google Scholar, Web of Science, Scopus and ORCID have established comprehensive online repositories that have recorded historic to present-day collaborative researcher interactions, characterising research organisations, their research partners and research outputs. I argue that these online repositories provide a significant opportunity to develop and use "big data" driven tools that combine advances in machine learning, data mining and social network technologies, to deliver research tools for the active support of research collaboration and enhanced network intelligence within the research management environment.

In this technical paper I introduce and evaluate a research collaboration proof-of-concept toolkit as a working example of a data-driven solution that utilises widely available online repository data sources to fill this research organisational gap. The overarching goal of my toolkit is to capture the collaborative network patterns associated with researchers from research organisations and their research partners who have collaborated together in the past, using these historic patterns to identify researchers who should be collaborating though are not. My toolkit thereby learns and visualises these patterns in the form of co-authorship, research expertise, publication vehicle preferences (conferences and journals) and affiliations to inform and direct organisational action for identifying and supporting these researchers who ought to collaborate or co-author, though have not yet done so.

Overall, my toolkit aims to provide four core functionalities: online researcher profile extraction and disambiguation, researcher expertise topic modelling, link prediction-based research collaboration recommendations and research network visualisation. As a proof-of-concept, the toolkit architecture is currently a sequential collection of fifteen Python and R Language scripts, with future implementations intended to cement these backend functionalities into an open-source Rshiny application, leveraging R's *Reticulate* package to deliver a front-end, user-friendly and interactive server-based interface that could remotely provide these four core functionalities to non-technical research organisation users.

The toolkit integrates numerous open-source research collaboration and network software applications into one streamlined accessible pipeline, creating a potential platform for research organisations to collectively map and integrate research centres in an organisation with their immediate, regional, national or international research context.

In evaluating the performance of my toolkit on a small sample of nine cancer researchers from the University of New South Wales Cancer Clinical Academic Group, I found that my four key functionalities perform as intended. With promising results in: correctly extracting, disambiguating and representing user-organisation researcher publication profiles *(average F1-score=85.7%)*, implementing human-interpretable topic-modelling of research publications and allocating modelled research topics to researchers, delivering researcher collaboration predictions and recommendations based on predicted graph-trained and sampled research collaborations *(AUC=98%)*, and finally delivering useful single-layer and multi-layer research collaboration network visualisations. However, in providing support for the utility and practicality of implementing such a

toolkit, there remains significant work required in its deployment and delivery for a more generalisable and user-friendly service.

In this technical report therefore, I define and explore this research opportunity I have identified in utilising these online research data repositories and evaluate the performance of this proof-of-concept toolkit. I also define and outline the future steps required in deploying my data-driven research collaboration open-source toolkit for the potential future use of the research community.

Executive Summary Post-Script

With the on-going humanitarian crisis of the COVID-19 pandemic, the global community and the research world has adapted to the need for minimal human contact to reduce the transmission of the infectious disease. In doing so, research organisations and researchers have showcased the ability of research collaboration to function outside of physical spaces, with research activities and interactions moving beyond geographical and concrete settings to digital platforms. I believe these online research trends will continue to be a substantial feature of research collaboration and research organisational environments, with increasingly remote work shifting research interactions to increasingly larger digital spaces. I suggest this digital research environment will likely create enhanced potential to provide a wider scope for research 'big data' capture and thus significant potential to increase the utility, scope and effectiveness of data-driven tools such as my proof-of-concept research collaboration toolkit within these organisations.

2.0 Introduction

Research and development (R&D) in universities and other research organisations are the engines of innovation, technological advancement and the production of new scientific knowledge and resulting improvements in well-being and wealth (Bozeman, Fay & Slade, 2013). There is a considerable onus on research organisations to maximise their R&D performance by measuring, stimulating and supporting R&D activities in the pursuit of both publication metrics and real-world impact from research activity (Bozeman, Fay & Slade, 2013). Of the many elements contributing to R&D performance, the microcosm of individual research collaborators within and between research institutions and their collaborative interactions plays a critical role, providing the foundation for R&D productivity within these organisations and collectively making up the global collaborative social 'web' or research collaboration network in which modern science and researchers are embedded.

Research collaboration is thus an important outcome within research organisations as both a measure of research output in the form of knowledge production through research publications, citations and translation or implementations, and secondly in the form of wealth and resource generation in terms of patents, new technology, business start-ups and organisational profits originated from researcher collaborations (Bozeman, Fay & Slade, 2013). There is an abundance of evidence for the support of interdisciplinary and interorganisational collaboration in the research setting, with evidence for increased innovation through shared expertise, ideas and resources, reduced university expenditure and increased academic productivity in the creation of new scientific knowledge and developments (Fonseca et al, 2016) (Lee & Bozeman, 2005)(Bozeman, Fay & Slade, 2013). Despite these efforts in supporting interdisciplinary and interorganisational research collaboration amongst research institutions,

these initiatives tend to run up against pervasive bottom-up structural forces which shape research collaboration social networks from within, typically with homophilic behaviours – that is, those characterised by an attraction towards similarity or homophilia in research at the individual and organisational level – favouring collaboration occurring within institutions and departments, with those individuals or organisations who share similar interests or specialities and with those already having worked with many collaborators. Wang & Zhu (2014) assert these behaviours influence the evolution of research collaboration networks, resulting in 'isolated' communities within and between research organisations, resulting in a strong 'siloing effect' of organisation resources and expertise (Tight, 2014)(Wang & Zhu, 2014).

In the last decade, a lot of research into the analysis and visualisation of research collaboration networks has occurred, and there have been significant improvements in the accessibility of data on research activity and outputs, particularly in data collected by online research bibliographic repositories. One of the motivations for the work described in this dissertation is that a significant opportunity would appear to exist for research organisations in bringing network-based research methodologies and online bibliographic data out of the research domain and into data-driven toolkits to inform efforts in supporting and shaping research collaboration networks within and between these organisations.

This report documents the development of a proof-of-concept toolkit to harness some of these advancements in research network and machine learning technologies. We propose a six-step integrated pipeline architecture for the synthesis of 'static' research organisation databases with freely-available online research repositories, leveraging concepts and goals derived from research collaboration link prediction, research expertise classification, research database author disambiguation, research expert finder systems and research network visualisation, to provide functionality and insights that help inform support efforts for researcher collaboration. In particular the toolkit can serve to iteratively update researcher databases, characterise researcher expertise based on their outputs, provide research collaboration recommendations and visualise the local context of partnered research organisations, research expertise, prominent researchers, publication vehicle and journals and, overall, the research organisations local research collaboration network. This may facilitate a more active but better targeted institutional role in support of research efforts and research collaborations.

3.0 Existing Resources and Methods

With the rise of the electronic age, medical and scientific databases such as PubMed, Google Scholar, Web of Science and Scopus, in addition to researcher profile databases such as ORCID, have over the past decade enabled the establishment of online repositories and services objectively recording the collaborative publication relationships between researchers throughout science and academia. These relationships are primarily represented by formal academic relationships such as co-authorship on published papers, and networks of publication citations. This has led to growing efforts to understand these networks and the social links and global collaboration trends they represent, in order to expand and promote research collaboration, expertise exchange and research diversification throughout the research milieu (Falagas et al, 2008)(Lee & Bozeman, 2005)(Fonseca et al, 2016).

There is an abundance of research into the use of these collaborative research data sources as a means of studying research collaboration. The literature is too large to survey completely here. In this report we focus on the technical approaches of research collaboration network visualisation, network statistical analysis and link prediction; predominantly basing these methods on co-authorship publication and citation relationships (Yan et al, 2018) (Sun et al, 2015) (Deng et al, 2008) (Zhou et al, 2019) (Sun et al, 2011) (Miller et al, 2014) (Silvia et al, 2013) (Afzal & Maurer, 2011) (Fagan et al, 2018) (Kang & Coppel, 2015) (*Sun et al, 2011) (Luo et al, 2014).

In terms of the provision of data-driven tools for the active support of research collaboration within research organisations specifically, three clearly segmented areas of research and implementation were identified in the literature, these being; finder systems for expert recommendations, research paper recommendation systems for suggesting suitable papers to researchers based on their research interests, and lastly, research collaboration link prediction studies for developing methods in predicting new research collaborations between researchers who have not yet collaborated. The methods described in this report draw on elements and functionalities from these three domains, but with an emphasis on the last (Afzal & Maurer, 2011) domains (Afzal & Maurer, 2011) (Haruna et al, 2017) (Li et al, 2018) (*Sun et al, 2011) (Zhou et al, 2019).

4.0 Methodological Pipeline

Our proposed research collaboration toolkit consists of *six technical stages* making up the complete pipeline framework and being intended as a proof-of-concept for laying a foundation or guide for future research collaboration tools to fill this perceived gap. These six pipeline stages are: *researcher data extraction*, *author disambiguation* (allocation of the correct papers to each author), *researcher profile consolidation* (through network comparisons), *expertise representation* (using topic-modelling), *link prediction* (for research collaboration recommendations) and finally, *network visualisation (*through the application of homogeneous or single-layer network visualisations and secondly multi-layer or multi-node and relation-type research collaboration network visualisations).

Overall, these six stages produce a four-fold core functionality for our toolkit – *illustrated in Figure 1* – providing a pipeline for research organisations to iteratively update their researcher databases with researchers latest publications through pipelines connecting our tool to live online bibliographic repository APIs (namely *Scopus*, *PubMed*, *ORCID* and *Web-of-Science*), modelling researcher expertise through topic-modelling of researcher publications, offering research collaboration recommendation functionality for finding unassociated researchers who should collaborate, and finally providing functionality to visualise and analyse these research collaboration networks through both single-level (homogenous) and multi-level (heterogenous) network visualisations.

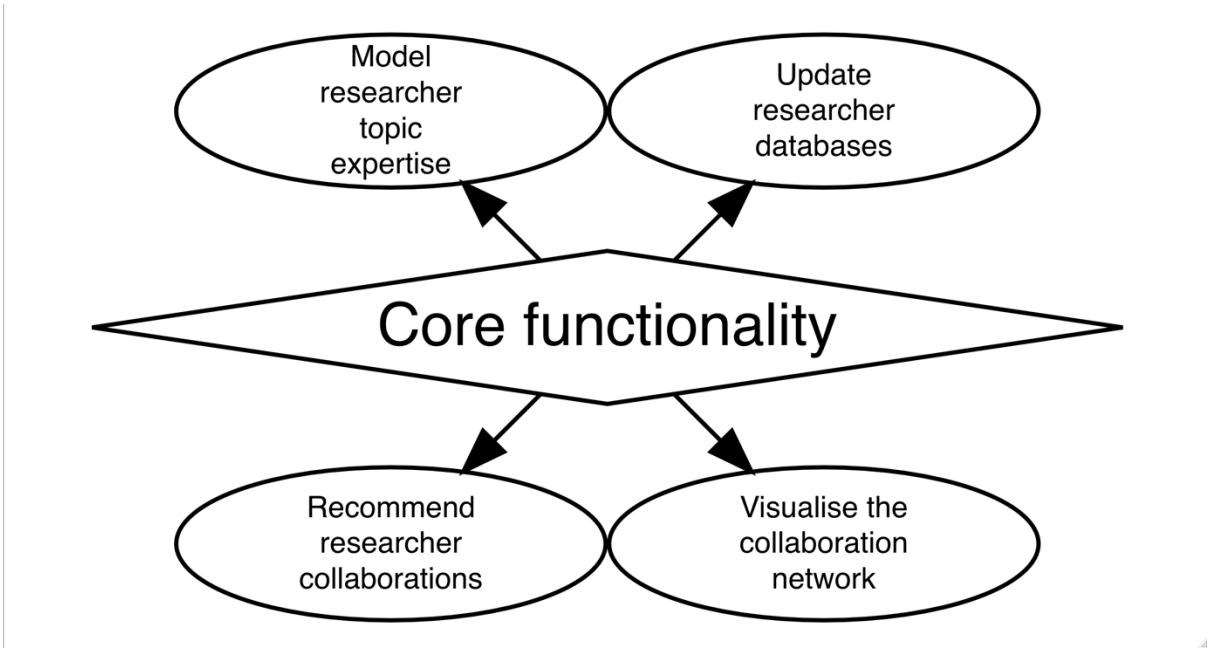

Figure 1 – Four-fold functionality of proof-of-concept toolkit

4.1 Research governance

The proof-of-concept results reported here use data for nine UNSW researchers, whose names were provided by one of the supervisors of this dissertation, Dr Tanya Ward, who sought volunteers from amongst her colleagues in the UNSW Cancer Clinical Academic Group. After consultation with Dr Ward and Professor Mark Ainsworth, both of whom advise the UNSW Faculty of Medicine on researcher metrics, it was decided that Human Research Ethics Committee oversight was not required for this project, provided that no comparative performance metrics were calculated and that the names of the researchers were obfuscated. The reason for this is that all data sources are publicly available (including the unobfuscated names of UNSW researchers), and the investigations reported here are entirely in line with work which is carried out routinely within the Faculty of Medicine (and other Faculties), without ethics oversight.

4.2 Stage 1: Researcher Extraction from Bibliographic Database Archives

The first stage sets the foundation for the research collaboration tool, *data extraction*, determining the quality, quantity and coverage of the publication data extracted for each researcher – or as we will now refer to them as ***'primary researchers'*** – and the boundary of effectiveness and variability in using our pipeline tool to the extent that unique researcher database IDs are provided for each primary researcher by the research organisation, and secondly the researcher coverage of their publications over our four chosen online researcher bibliographic repositories for data extraction: ***PubMed***, ***Scopus, ORCID*** and ***Web-of-Science***.

The choice of these four bibliographic data archives is based on maximising content coverage and leveraging practical utility, particularly in terms of availability of APIs and complementary software packages for efficient and convenient extraction, with the inclusion of Scopus incorporating wide coverage of articles in scientific fields published after 1966, Web-of-Science complementarily covering publications prior to 1900 and PubMed

most effective in covering content within medicine and biomedical sciences (Falagas et al, 2008) (Luo et al, 2014). Other data sources, such as Google Scholar were considered, however, are generally thought to be less reliable though more obscure in content coverage (Falagas et al, 2008). Further development of the toolkit, beyond the scope of this dissertation, could usefully add Google Scholar as a source.

Overall, our extraction method is heavily dependent on the availability of unique researcher identifiers. Primarily, for each respective bibliographic archive a unique database identifier is used for extraction, being associated with each online database for each researcher (PubMed ID, Scopus ID, ORCID ID and Researcher ID respectively). Secondarily, ORCID IDs from ORCID (an online research organization for establishing universal researcher IDs and researcher profiles) are used as a 'back-up' ID for PubMed, Scopus and WOS pipelines in addition to ORCID primarily, with unique identifiers being obtained from research profiles provided by the user-research organisation. As an example of the functionality of the tool, this technical paper uses a researcher cohort provided to us by the University of New South Wales (UNSW) Cancer Clinical Academic Group (CAG), consisting of 9 selected primary authors specialised in cancer research from this research organisation, having all unique IDs available and whom we will use to evaluate and present each stage and output of our proof-of-concept tool.

The extraction process uses the '*R Statistical Language'* following *Figure 2*, with each online researcher database having its own separate R pipeline script feeding into a final merging script, producing an aggregated but potentially ambiguous dataset for all primary authors and co-authors – or as we will now refer to non-primary researchers (or those not associated with the primary research organisation) as ***secondary researchers*** – relying on each corresponding bibliographic R package; *Rorcid, Rscopus, rwos* and *rpubmed;* to extract researchers bibliographic data from our online archives APIs (R Core Team, 2017). By *ambiguous* we mean that the extracted data may contain papers and articles by authors of the same name as the primary researcher, but who are, in fact, a different person. This and related problems are discussed in more detail in the next paragraph.

During extraction, all papers associated with each researcher's unique researcher identifier are extracted, 'scraping' all available researchers publication textual information (published titles and abstracts), publication vehicles (journals or conferences), year of publication, digital object identifiers (DOI) , co-authors and organisational affiliations associated with each unique researcher identifier and publication. These datasets are then aggregated and merged for both primary and secondary research publication profiles, allocating publication records to each research author based on pre-processing of names through basic removal of all punctuation and normalization of names to complete surname and incomplete first name initial blocks. This introduces two challenges associated with exploiting online bibliographic data sources, namely *author name homonymy* (different researchers sharing the same name in publication databases) and *author name synonymy* (researchers having different names in publication databases) (Fucella et al, 2016) (Backes, 2018). To deal with researchers with the same surname and first initial, author disambiguation is employed as a solution in phase two, involving clustering of separate researcher profiles under the same name into distinct authors, with the performance of this algorithm being evaluated on our researcher cohort using weighted accuracy (F1-score) as our performance metric (Fucella et al, 2016). For *author name synonymy*, we have chosen not to employ the mainstay method of

blocking and *record linkage* to resolve this issue – as is often the case when dealing with a general author archive search over online bibliographic data sources – as our extraction method is informed entirely upon unique researcher identifiers associated with established online researcher profiles and based on APIs with standardized name formats for extraction amongst all four bibliographic archives. Rather, we have implemented only basic incomplete name matching to aggregate and capture surname and first initial for all authors, with an assumption being made that this will reduce *author name synonymy* at the cost of significantly increasing the occurrence of *author name homonymy* and thus influencing the rate of false positive publication extraction (Fucella et al, 2016) (Backes, 2018).

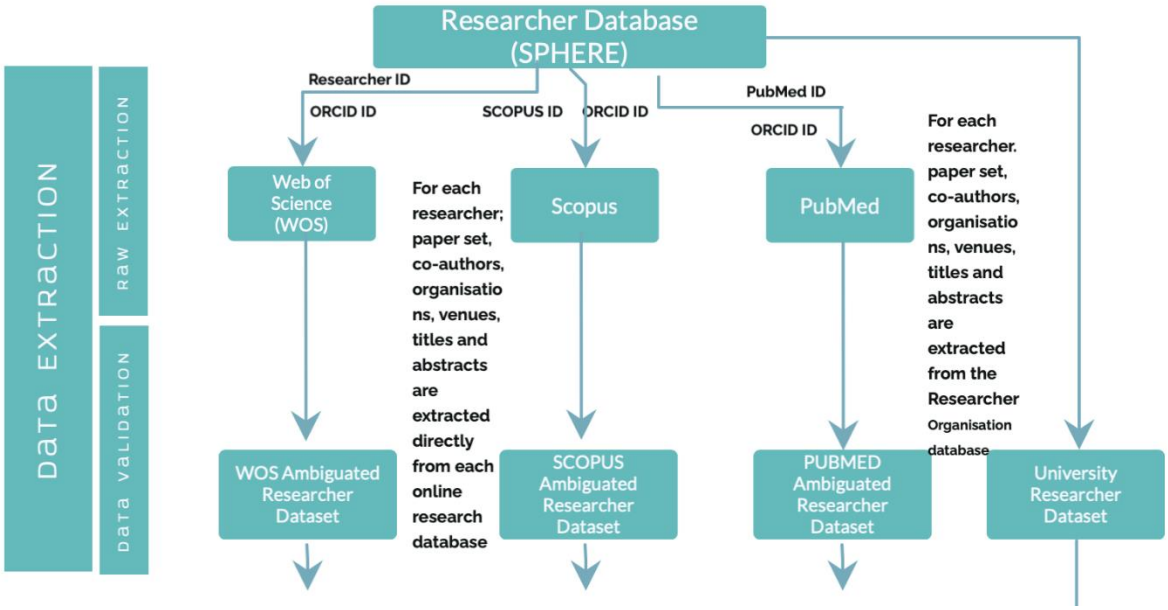

Figure 2 – Stage 1 data extraction pipeline

4.3 Stage 2: Researcher Publication Disambiguation

Researcher or author disambiguation addresses the key challenge of author name homonymy in bibliographic data analysis, with two key considerations to be taken into account in solving this problem. Firstly, how best to represent research papers effectively making use of all biographical information for a group of ambiguously-named researchers (***'right papers'***), and secondly how to determine the assignment of these representations to the correct author (***'right researcher'***) (Backes, 2018) (Fucella et al, 2016). To solve this first problem, we implement a ***python*** based solution introduced by Xu et al (2018), employing a network-embedding approach to learn researchers name-ambiguated paper representations, and secondly, an adapted clustering methodology to allocate the right set of papers or ***'right papers'*** amongst ambiguous researchers who may share common surnames and first initials with one another (Rossum, Guido & Drake, 1995). We chose this disambiguation method for being one that was accessible, efficient, and effective – outperforming existing evaluated methods by approximately 7% to 50% (Xu et al, 2018).

As shown in *Figure 3*, using the extracted researcher data from stage one, five separate graph-networks are constructed based on five distinct network relationships derived from the researchers extracted biographical information (co-title, co-abstract, co-affiliation, co-venue and co-author), a global network-embedding and

network coarsening procedure is then applied to these five networks, resulting in a condensed global network embedding which reduces these five network dimensions into a global network vector representation for input into two competing clustering algorithms; these being HDBSCAN and affinity propagation clustering algorithm (AP) (Xu et al, 2018).

Xu et al (2018) then uses the SD index as a metric to impose a decision threshold over the results of both of these clustering algorithms, providing an adaptive leverage for utilising the competitive behaviours of both methods; with HDBSCAN tending to cluster data into fewer clusters and conversely AP tending to cluster data into many clusters (Ricardo et al, 2013)(Frey & Dueck, 2007). In our case, we hypothesis it would be undesirable utilising AP's bias towards a default preference for researchers being considered as multiple researchers, thereby often dividing these researcher's publications into multiple clusters. Thus, as our research cohort and methodology is inherently different in the degree of potential researcher ambiguity attributed to each author, due to a reliance on unique identifiers for extraction alone compared with Xu et al's (2018) more generalised name-only bibliographic extraction method, we hypothesis using HDBSCAN alone may increase performance in the identifier-dependant context of our proof-of-concept tool. This was born out in practice, and thus we thereby implemented HDBSCAN alone; preferring fewer clusters and placing emphasis on avoiding increased sensitivity towards maximised splitting of all researchers publication profiles on profile-specific proportional variation that may likely be due to the common textual errors and inconsistencies characterising bibliography data sources rather than indicating homonymous authors sharing the same name.

Our method, therefore, adapts Xu et al's (2018) approach by relying only on HDBSCAN clustering as a means of tolerating this 'noise', dropping AP clustering and Xu et al's (2018) decision threshold. For most university settings, this use of HDBSCAN alone is likely to be an effective algorithm for balancing the identification of outlying homonymous researchers sharing the same name, whilst tolerating 'noisy' unique researcher profiles within our different context of unique identifier-dependant publication extraction.

4.4 Stage 3: Researcher Profile Consolidation

Having disambiguated each potentially-ambiguous researcher profile into clusters representing distinct researchers and having allocated each researcher's '***right papers***' between authors sharing the same name in stage two, the next stage is the identification and assignment of a correct unique paper cluster to each primary researcher in the scenario where there are numerous researcher clusters created by our disambiguation stage (Fucella et al, 2016). Our approach to this problem uses stage three as an adapted final-step in Xu et al's (2018) disambiguation algorithm, determining which disambiguated researcher publication cluster best represents the organisation's primary author (being disambiguated), with this requiring a means of comparison between a researcher's 'ground-truth' organisation-held researcher publication record as a reference against each corresponding disambiguated paper cluster returned for each researcher.

To achieve this goal, we apply '*graph edit distance' (GED),* a form of inexact or error-tolerant graph matching and network comparison often applied in comparing the distance between two different graph networks (Zeina et al, 2015). Presenting as the second-half of our stage-two to three pipeline as shown in *Figure 3*, the algorithm

works through finding the best (minimum) set of transformations between one graph network and another by means of graph editing operations, including inserting, deleting and substituting graph vertices and their edges, finding the minimum-cost sequence of possible transformations and using this as a distance metric between two graph networks. It is therefore analogous to the Levenshtein edit-distance for string comparisons. We adopted Fize et al's (2018) implementation for our approach (Fize et al, 2018). Our first graph in the comparison is the reconstruction of Xu et al's (2018) five relationship networks (co-author, co-title, co-abstract, co-venue and co-organisation) for each disambiguated paper cluster, with our second reference graph consisting of the corresponding five relationship networks based on the organisational researcher database for each researcher provided by the user-research organisation (Zeina et al, 2015) (Tantardinin, 2019). We therefore create five-layer 'reference networks' representing the organisation's ground truth publication profile for the primary researcher in question, calculating an average GED over each cluster set of each specific publication layer network (e.g. cluster co-author network) against the corresponding relationship network of the five-layer reference network (e.g. reference co-author network), taking the publication cluster with the smallest GED average from the reference network over all five-network relationships as our desired similarity indicator between a researcher publication cluster to our primary researcher of interest.

Having assigned the correct paper cluster to the '***right researcher'***, we next undertake a basic name matching procedure on each primary researcher's ORCID researcher profile using researcher ORCID IDs. This involves confirming that a researcher's surname and first initial are an exact match with the corresponding ORCID profile record, providing support for adding ORCID profile publications as true publications to extracted and disambiguated researcher profiles. With this confidence inferred from the personally-curated nature of ORCID profiles set up by and managed by individual researchers or their organisations. We thereby assume that these publications are correct and add these to our disambiguated research paper cluster, representing the final extracted bibliographic researcher publication profiles for each researcher and theoretically allowing for updating of university researcher publication databases. We will evaluate and discuss our combined stage one to three extraction and adapted disambiguation methodology in our *Results* section, applying this core functionality of our tool to our nine UNSW primary researchers.

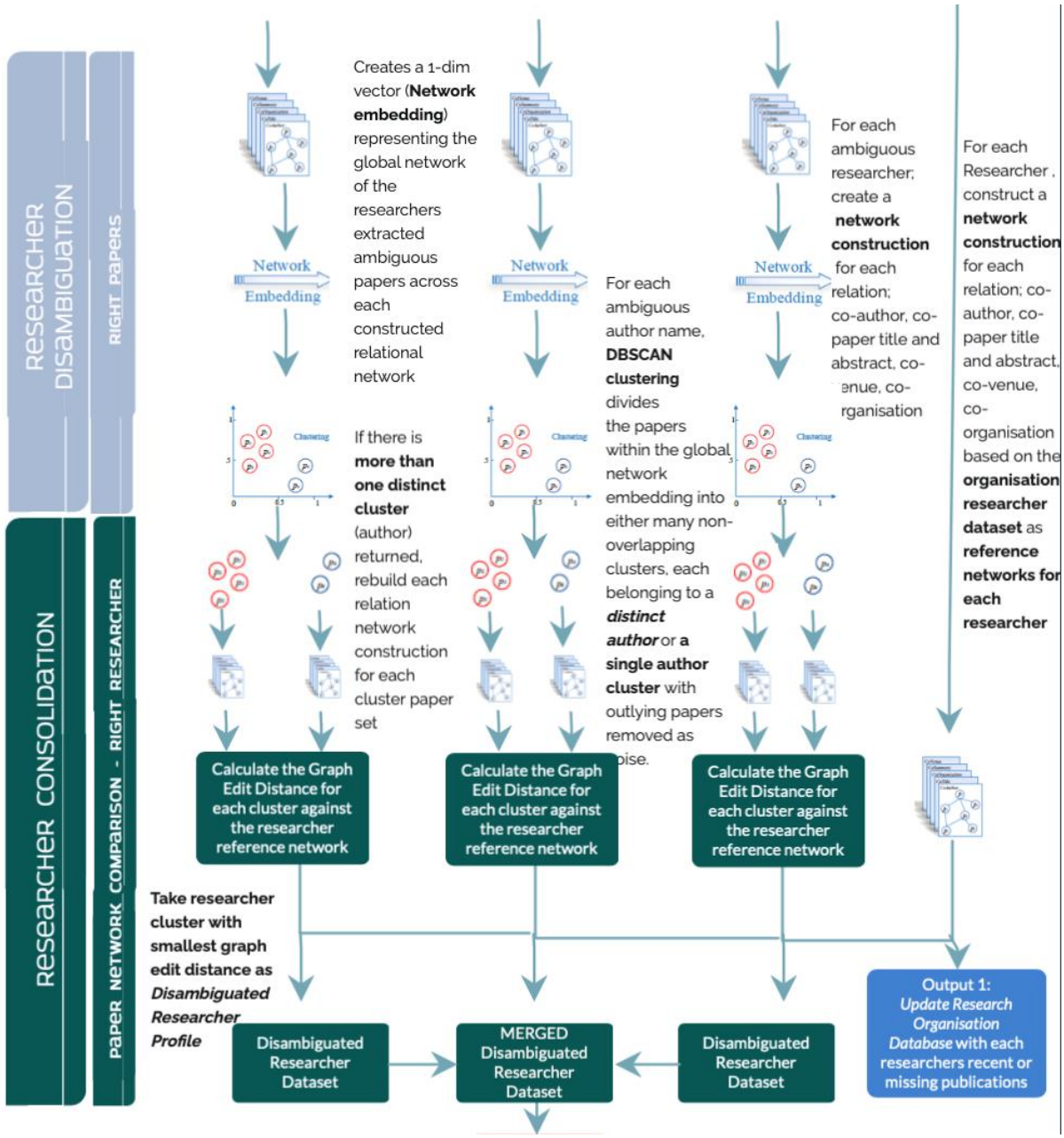

Figure 2 - Stages 2 to 3 research disambiguation and researcher consolidation pipeline

4.5 Stage 4: Representing Researcher Expertise through Topic Modelling

Stage four is concerned with the representation of researcher expertise, with our researcher's previous publications gleaned from stages one to three, providing a signal and means of modelling the latent research topics of each researcher through the application of a text mining approach in the context of our organisation's known research speciality; in this case, cancer research (Momtazi & Naumann, 2013) (Katsurai et al, 2016). Using each researcher's extracted publication information (titles & abstracts) as 'bag-of-words' vectors for input, our method applies a pre-trained Latent Dirichlet allocation (LDA) topic-model – a generative three-level hierarchical probabilistic topic model, using the implementation derived from the python topic-modelling package, '*Gensim*', training this model on a corpus of 10,000 PubMed 'cancer-related' publications, as published and validated by Zhu, Lyu & Ji (2020).

Our LDA approach is centred upon the estimation and modelling of latent topic distributions underlying the literature of the user-research organisations specialisation and publications, in this case cancer-related research, capturing topics as word probability distributions over our corpus of 10,000 documents from our PubMed cancer literature training corpus, with each publication document being topic modelled as a finite mixture of words over an underlying set of probabilistic word topic distributions, and with each topic conversely being modelled as an infinite mixture of topic words over an underlying set of topic probabilities – with these topic probabilities then providing a mechanism for capturing each of our researchers latent topics of expertise in a subsequent 'test' and topic allocation phase of our unseen primary and secondary researcher publications (Blei, Ng & Jordan, 2003).

As shown in *Figure 4*, our pre-trained LDA is positioned as a pre-defined central component of our tool for modelling the organisational distribution of cancer research expertise. Having disambiguated our cancer researchers' publications in stage two, our method applies this trained LDA to determine and assign the underlying cancer research topics present within each researchers 'unseen' publications, thereby functioning as the test dataset for our trained LDA model. In training our pre-trained model, we use '*coherence'* as our guiding metric to ascertain the optimal number of research topics within the PubMed training corpus that maximises human topic interpretability. Topic coherence is an alternative to other topic model evaluation metrics such as perplexity, focusing primarily on semantic human interpretability of topics rather than optimising for more probabilistic in-sample statistical predictive properties (Mimno et al, 2011). We chose this metric to maximise transparency and interpretability for end-users in our expertise representation stage, with topic word distributions modelled by our trained LDA requiring a key concluding step of manual interpretation by research organisation users to determine and assign discrete labels for each topic distribution as inferred by organisation specialists (Mimno et al, 2011). In other words, LDA does not suggest names for the topic clusters it finds – that remains a step requiring expert human input, at least in the toolkit pipeline as described here. However, we provide some tools to make this task easier.

In inferring underlying research topic labels over our researchers' publications for our cancer research organisation, we leverage the python package *pyLDAvis* to visualise the top-most salient words supporting each modelled topic distribution to users (Sievert & Shirley, 2014). The interactive functionality presents each topic's level of separation to users based on a 2-dimensional primary component analysis (PCA) visualisation (Sievert & Shirley, 2014). This facilitates research organisation staff to interactively interpret and infer the quality of topics and assign the appropriate topic label guided by this visualisation tool and their own understanding of their research organisations specialities. These labels, assigned to research topic distributions, are visualised as labelled topic nodes associated with each researcher in subsequent stages of our pipeline (Deng et al, 2008) (Kang & Coppel, 2015) (Afzal & Maurer, 2011) (Katsurai et al, 2016).

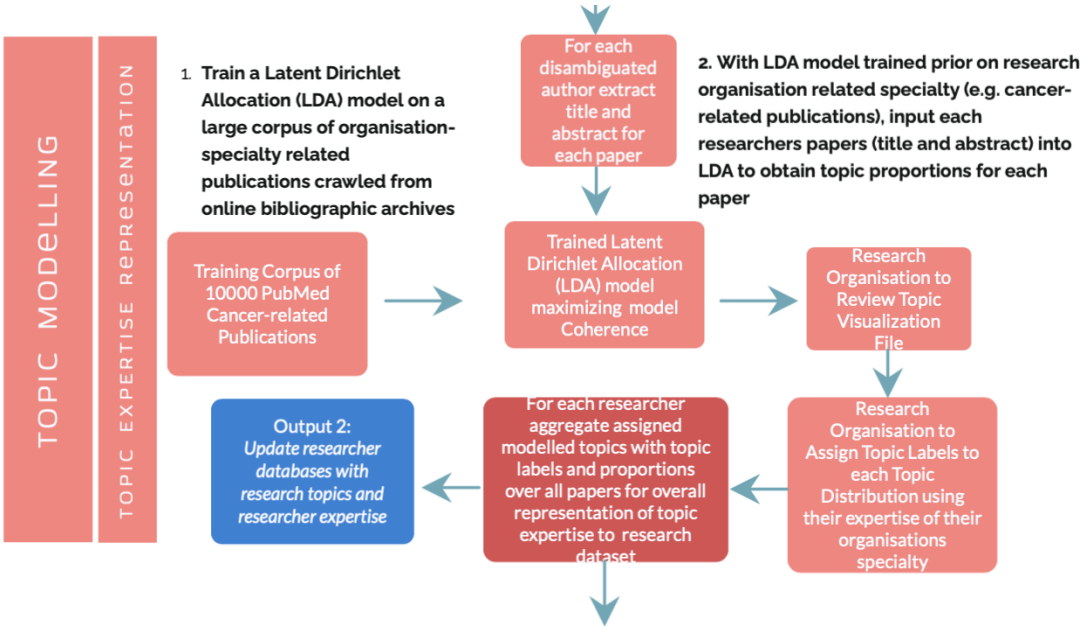

Figure 4 - Stage 4 research topic and expertise representation

4.6 Stage 5 Research Collaboration Recommendations utilising Graph Link Prediction

Stage five takes the unstructured bibliographic data extracted, disambiguated and topic-modelled in prior steps of our research collaboration pipeline, and uses this as input into a prediction task and subsequent recommendation system functionality to predict or recommend future (hopefully successful) researcher collaborations between previously unassociated researchers in a research organisation (or across organisations if the data spans several organisations). To do this we leverage recent graphical representational network embedding techniques to capture the rich and complex semantic and structural latencies of the bibliographic network encoded within our extracted cohorts publications; informed by the historic research collaboration interactions of the research community.

In leveraging these graph network representations to model research networks, link prediction is the main task, involving the prediction of edges representing a shared publication between two researcher nodes (Liben-Nowell & Kleinberg, 2007). This problem has been comprehensively explored over the last decade with initial approaches based on calculating descriptive network heuristics, often based on homogenous networks of one single node or edge type as an embedded representation of a network; such as co-authorship networks of researcher nodes and co-author edges representing pairwise co-authorship publications (Shakibian & Charkari, 2018). However, in reality, most social and information networks are heterogeneous or complex in nature, consisting of an exponential diversity of different types of entities and interactions and thereby requiring a more complex, multi-layered model of the social environment, one which acknowledges this heterogeneity of entities, relationships and pathways within and through research social networks to better predict new research collaborations (Sun et al, 2011).

Sun et al (2011) firstly introduced the definition of mixed network pathways or meta-paths on which current heterogenous network embedding approaches are now based, creating a conceptual framework and implementation through random walks between defined pathways of differing network edge and node types, that

provides global network vector embedding of mixed relational network pathways through a heterogeneous network. Furthermore, Sun et al (2011) introduces a network schema template to formally map the explicit set of node-to-edge meta-paths to be explored by random-walks within a heterogeneous network (Sun et al, 2011). Using these definitions and methods, our meta-network schema of mixed node and edge pathways (metapaths) between our nodes (author, organisation, topic, and venue [journal or conference]) and corresponding edges is defined in *Figure 5*.

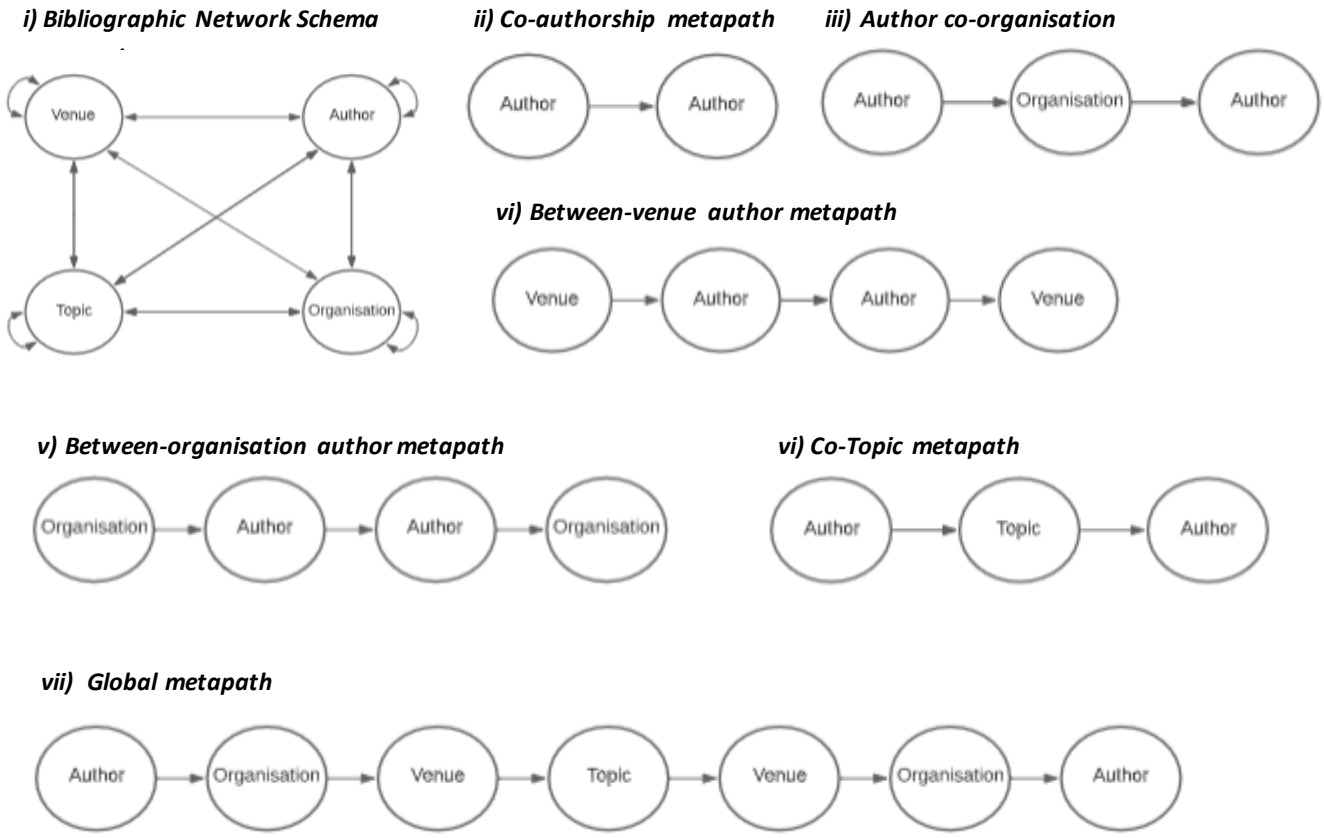

Figure 5 – Network schema and defined meta-pathways

Following our stage five pipeline in *Figure 6*, we construct a four-layer heterogeneous network consisting of our key node types (authors or researchers, venues, stage four modelled topics and organisations) and relational edge types; co-authorship, co-organisation, co-venue and co-topic and additional relation interactions between our four key entities defined in our meta-network schema (*Figure 5)*. We then employ an unsupervised heterogeneous network embedding algorithm – '*metapath2vec'* – to learn the low-dimensional vector representation of each node and it's neighbourhood context or node community through meta-path based random walks of our defined meta-pathways, embedding structural and semantic correlations captured within our distinct set of meta-pathways between our nodes and edges, and providing a global vector embedding of our heterogeneous network structure and the neighbourhood research context of organisations, topics, venues and co-authors interacting with each researcher. This vector representation is then used as input into a supervised binary classification task between researchers' embedded researcher nodes (Yuxiao, Nitesh & Ananthram, 2017). In contrast to earlier meta-path-based methods, the latent-space vector representations learned using *metapath2vec* can additionally model similarities between nodes without connected meta-paths through negative

sampling, providing a significant advantage to prior methods in capturing negative edge pathway instances and a more global representation (Yuxiao, Nitesh & Ananthram, 2017). The researcher node vectors produced are then processed using either L1 regularization or L2 regularization link operators, creating embedded link edge vectors with equal dimensionality to capture the co-authorship edge context between two researcher node embeddings and allowing a link prediction classifier to then learn the probabilities associated with each co-authorship link embedding and associated researcher embedded node pairs (Yuxiao, Nitesh & Ananthram, 2017). We followed CSIRO Data61's (2018) method and chose a logistic regression model for training and testing on these embedded link vectors, using this to predict the probability of new research collaborations or new co-authorship publications between two previously unassociated network researchers, taking the best link operator as a basis for model comparison with a more complex popular gradient boosting ensemble model – *LightGBM* – to further evaluate the trade-off between complexity in using different binary classifiers, and evaluating this method with Area-Under-The-Curve (AUC) as our performance metric.

The procedure of our heterogeneous network construction and Metapath2vec embedding methodology are implemented by the '*StellarGraph'* package in python, whereby they have adapted an approach based upon Yuxiao, Nitesh & Ananthram's (2017) heterogeneous link prediction task, laying down the functionality for building our heterogeneous bibliographic network, undertaking splitting of the raw graph dataset into training, model selection and test graph datasets through random sampling, implementing Metapath2vec embedding, applying link operators separately to node embeddings and training/testing these on two respective logistic regression classifiers (CSIRO's Data61, 2018) (Yuxiao, Nitesh & Ananthram, 2017). Adapting their implementation, we started from our heterogeneous network dataset, splitting this into a learning set (training set) and a prediction set (test set) through random positive and negative edge sampling. The learning set represents the current state of a research organisations collaboration network, laying the basis for where predictions are made. Conversely, the prediction set represents the future evolution of the organisations research collaborations, or specifically the list of new co-authorship edges or removed sampled edges between researchers to be predicted (Sinha, Cazabet & Vaudaine, 2018).

Our method, following that provided by CSIRO's Data61 (2018), uses a single, static graph without temporality as opposed to a dynamic temporal graph, thereby ignoring the order of edge apparition or year of researcher's publications. This is a useful simplification for this report, but temporality should be included in a production implementation. For this static context, random sampling is firstly used to remove a default 10% of both positive and negative co-authorship edges during an iterative edge splitting of the original graph into three elements; a testing set of sampled and removed test edge co-authorship labels, corresponding test co-author edge embeddings and remaining reduced graph of all nodes and unsampled edges. Secondly, another splitting procedure is then undertaken to further reduce the processed graph into a 75:25 training split of respective training and model selection/validation co-author edge embeddings and corresponding co-author edge labels, with the original heterogeneous graph ultimately reduced into a 'current state' learning set (training and model selection) and 'future state' prediction set (test set) for representing future evolutions of the research collaboration network in terms of new research collaborations or co-authorship edges, thereby avoiding data leakage between training, model selection or validation and test datasets in regard to target co-authorship edges (Sinha, Cazabet & Vaudaine, 2018). Finally, using AUC as our performance metric we compare both L1 and L2

link embedded models in model selection, and use the best performing link operator-transformed and validated logistic regression model to evaluate the effectiveness of our approach in predicting research collaborations of our cancer researchers against a similarly trained Light Gradient Boosted Model (LightGBM) of higher complexity (Ke et al, 2017).

The functional output of this stage therefore, can provide research organisations with a list of recommendations for new research collaborations between pairs of researchers who would likely be successful in co-authoring publications together but who have not published together in the past, basing these decisions on the indirect and direct historic correlations between differing venues (journals and conferences), organisations, modelled research topics and co-authors interacting between and around primary researchers within the organisations research collaboration network.

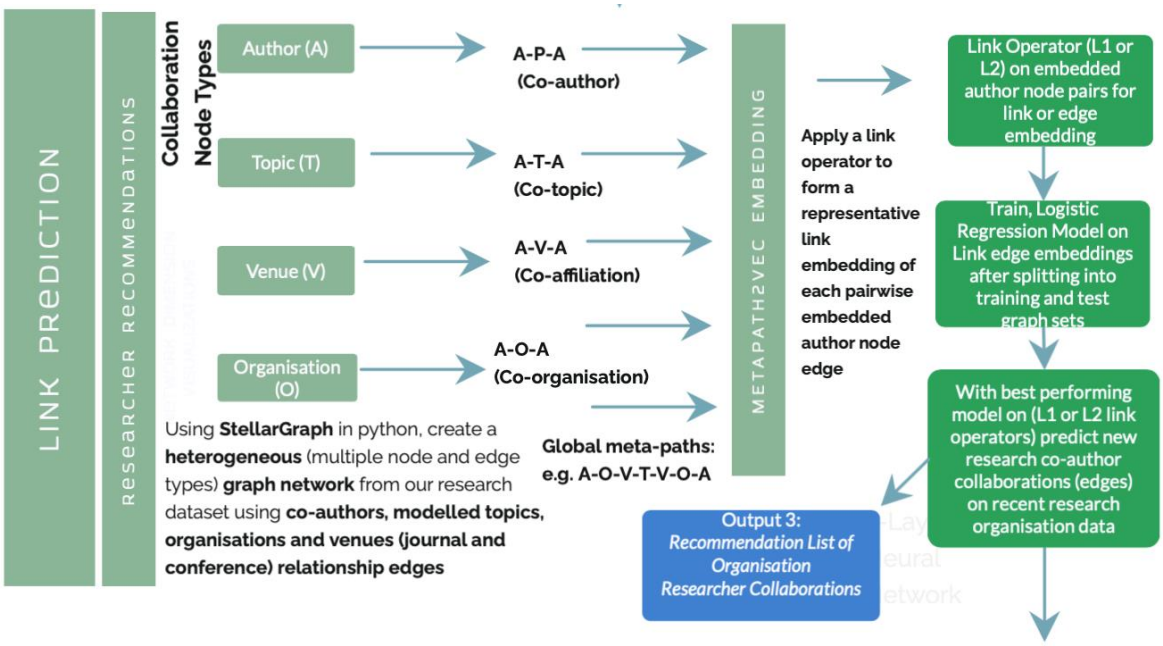

Figure 6 - Stage 5 link prediction and research collaboration pipeline

4.7 Stage 6: Research Network Visualizations

Stage six concludes our pipeline and proof-of-concept toolkit by providing research organisation users with a visualisation functionality to gain insight into their research collaboration network from two network vantage points; firstly through homogenous or single-layer static visualisations and secondly through heterogenous or multi-level interactive network visualisations. Firstly, for homogenous network visualisations, following *Figure 7*, we leverage the open source *Gephi Visualisation software* and R package *MuxViz* for producing homogenous network visualisations of the co-authorship network of the organisation's research collaboration network (Bastian et al, 2009). These networks are filtered and represented using two representations: node degree (the number of edges intersecting a researcher node) and, secondly, edge weight normalisation using maximum normalisation of edges, with these calculated for both heterogenous and homogenous networks. Users can manually bring outputted Gephi files into the *Gephi* environment allowing users to use *Gephi's* functionality and graphical-user interface of workspaces to manually explore, analyse and visualise the organisations central

homogenous network, assessing community modularity, spread and influential individuals through the lens of the co-authorship network (Bastian et al, 2009). Furthermore, a more custom and automated overview of the homogeneous network and associated recommendations from stage five is delivered using *MuxViz* for single-layer visualisation of the co-authorship network, with colour and size of researcher nodes specifying primary from secondary researchers and further highlighting those researchers involved in research collaboration recommendations.

Moving away from homogenous visualisations, stage six secondarily produces multi-layer interactive heterogenous network visualisations leveraging *MuxViz (*De Domenico, Porter & Arenas, 2014*)*. Having visualised in-depth the key single-layer homogenous co-authorship network of the organisation, users are able to further extend MuxViz functionality to produce interactive multi-level networks in the visualisation of the heterogenous research collaboration network of stage five – visualising four-layers (one for each node type; labelled modelled topics, organisations, venues[journals and conferences] and researchers) within the network and corresponding intra-relationship edges within each layer of a single node-type and their inter-relationships between each unique layer of each other node type, thereby producing an interactive view of the research multilayer independent network of the research organisation. MuxViz's multilevel network analysis and statistical functionalities can then be used on these data *(*De Domenico, Porter & Arenas, 2014*)*.

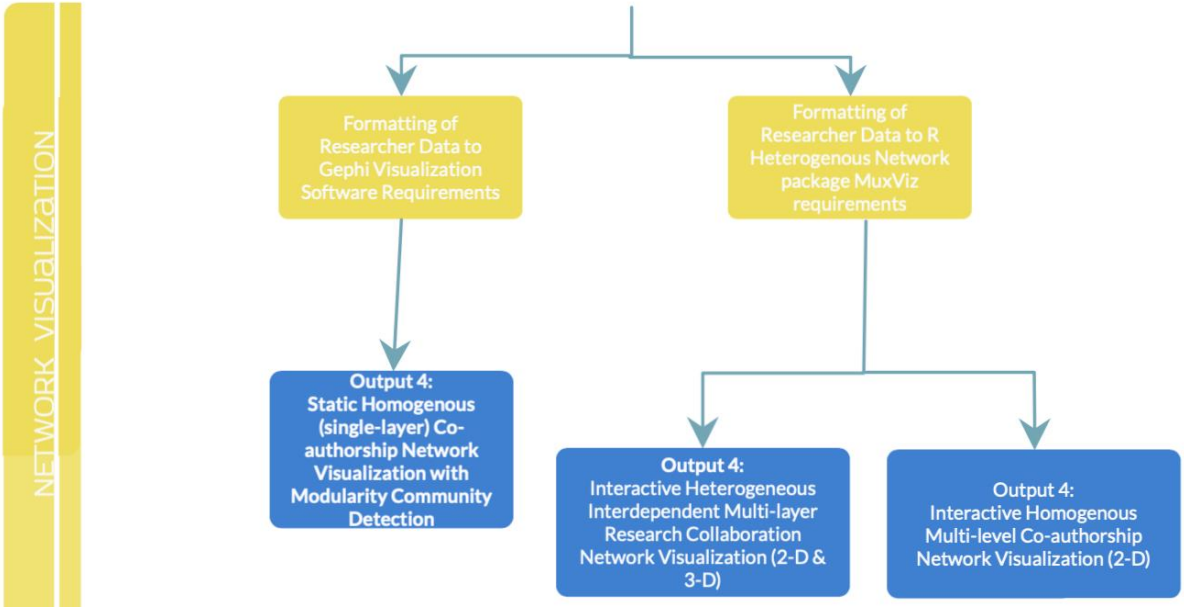

Figure 7 - Stage 6 network visualisation pipeline

5.0 Results

5.1 Output 1: Extraction, Author Disambiguation and Researcher Profile Consolidation

Output 1 encapsulates the result of researcher extraction, disambiguation, and research profile aggregation of stages one to three of our research collaboration pipeline, extracting researcher publication profiles from online bibliographic databases with the goal of automatically updating research organisation researcher databases. The performance of this data extraction and disambiguation process is critical to the functionality of our toolkit, laying down the foundation of adequate data quality, publication coverage and accuracy for downstream key database updating, topic modelling, network inference and visualisation tasks. Based on our nine primary researchers, our extraction and validation method was evaluated using confusion matrix statistics, comparing each researcher's account or '*known researcher publication profile (KRPP)'* of published publications as a gold standard reference of a researchers publication record against an '*extracted research publication profile (ERPP)'* formed utilising our extraction and validation method of stages one to three. We define true positive to be a researcher's publication extracted into the ERPP which is also in the researchers own KRPP. Conversely, false positives are defined as a publication extracted into the ERPP, though not contained within the researchers KRPP. True negatives are defined as publications not extracted by the tool into the ERPP and accounted for by the researcher in the KRPP (the gargantuan undefined number of research publications currently published). Finally, false negatives are defined as those publications found within a researchers KRPP though not extracted by our tool into that researchers ERPP. Due to the nature of our near-infinite true positive definition and open-world publication problem, we could not use true negatives and associated test measures such as accuracy and specificity in evaluating our tool. Instead we evaluate our extraction model's performance using *sensitivity* or recall, the ability of our tool to correctly 'classify' or extract a researcher's true publication profile over their research career, and *precision* or positive predictive value (PPV), representing the percentage of publications correctly extracted by the tool amongst our research cohort (Parikh et al, 2008).

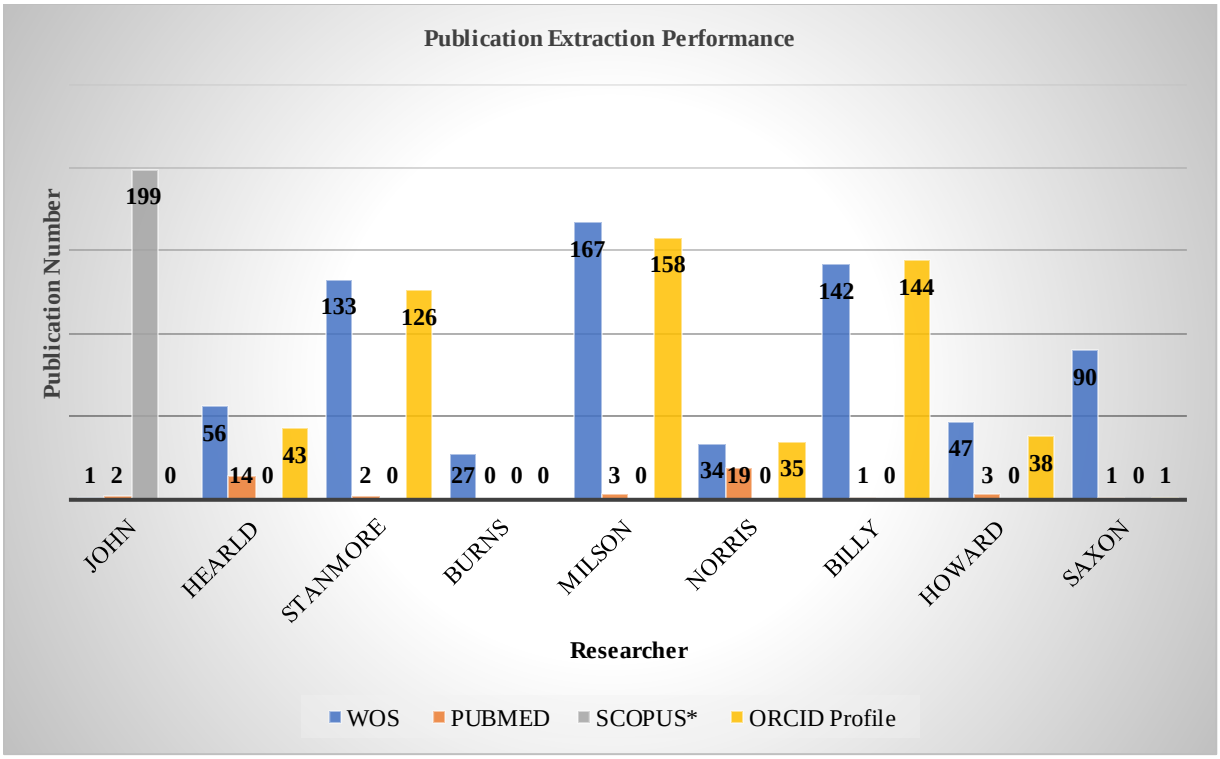

Figure 8 – Clustered histogram of researcher extraction performance

Researcher	WOS	PUBMED	SCOPUS*	ORCID Profile
JOHN	1	2	199	0
HEARLD	56	14	0	43
STANMORE	133	2	0	126
BURNS	27	0	0	0
MILSON	167	3	0	158
NORRIS	34	19	0	35
BILLY	142	1	0	144
HOWARD	47	3	0	38
SAXON	90	1	0	1
Grand Total	**697**	**45**	**199**	**545**

Table 1 – Table of researcher extraction performance

Figure 8 and *Table 1* above reveal the absolute number of publications extracted for each of our nine primary researchers utilising each bibliographic database extraction pathway. We can observe the two primary bibliographic databases providing most coverage for our researcher cohort are Web-of-Science and ORCID, with PubMed making a minimal contribution overall and Scopus only providing extracted publications for a single researcher – please see the *post-script* for further discussion of our Scopus results. Based on these results, we recommend that at a minimum a *Scopus, ORCID or Web-of-Science ID should be available for each researcher.

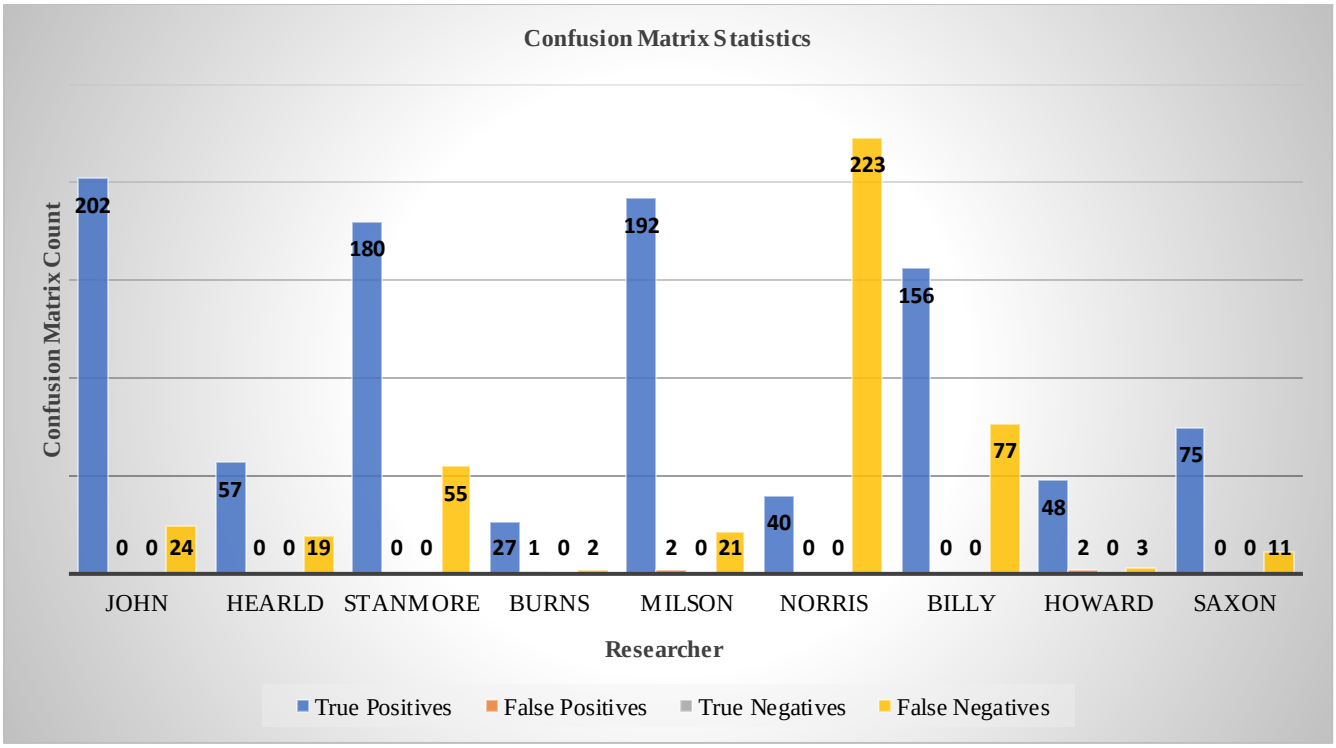

Figure 9 – Clustered histogram of confusion matrix statistics for each sample researcher

Researcher	True Positives	False Positives	True Negatives	False Negatives
JOHN	202	0	0	24
HEARLD	57	0	0	19
STANMORE	180	0	0	55
BURNS	27	1	0	2
MILSON	192	2	0	21
NORRIS	40	0	0	223
BILLY	156	0	0	77
HOWARD	48	2	0	3
SAXON	75	0	0	11
Grand Total	**977**	**5**	**0**	**435**

Table 2 – Confusion Matrix Statistics for each Sample Researcher

Figure 9 and *Table 2* provide a breakdown of our key diagnostic confusion matrix statistics, revealing that our method successfully extracted a significant proportion of true positives against false negatives, with the occurrence of only 5 false positives overall. It should be noted, that in utilising our disambiguation algorithm in stage two we used default parameters as specified by Xu et al (2018), thereby leaving space for hyperparameter tuning in optimizing our disambiguation performance and likely reducing false positive occurrences in future iterations of the tool.

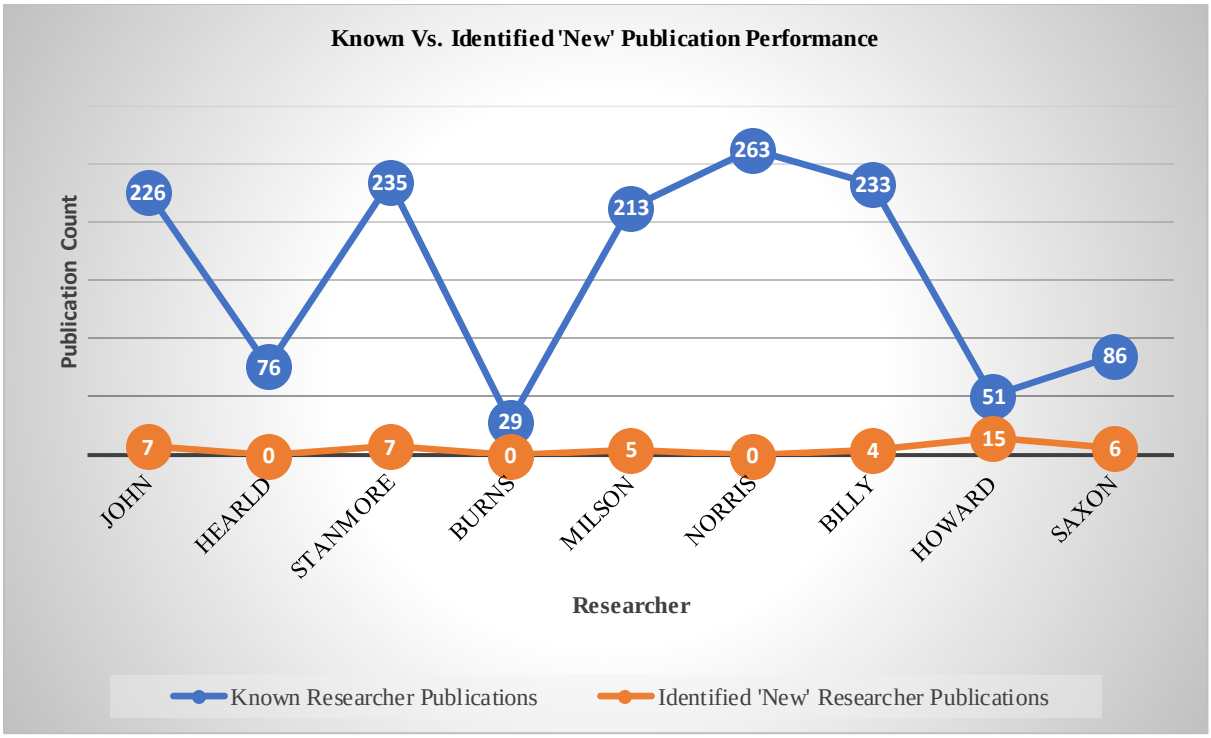

Figure 10 – Line graph of known vs. unknown or 'new' identified publications extracted for each sample researcher

Researcher	Known Researcher Publications	Identified 'New' Researcher Publications
JOHN	226	7
HEARLD	76	0
STANMORE	235	7
BURNS	29	0
MILSON	213	5
NORRIS	263	0
BILLY	233	4
HOWARD	51	15
SAXON	86	6
Grand Total	**1412**	**44**

Table 3 – Known vs. unknown or 'new' identified publications extracted for each sample researcher

Figure 10 and *Table 3* illustrate the amount of known researcher publications within each primary researchers KRPP against the amount of extracted and confirmed 'new' researcher publications identified by researchers as publications of which had not been included or were not 'known' were missing from their KRPP. This highlights a potential useful feature of our extraction pipeline in updating researcher databases with both known and previously unknown or missed researcher publications.

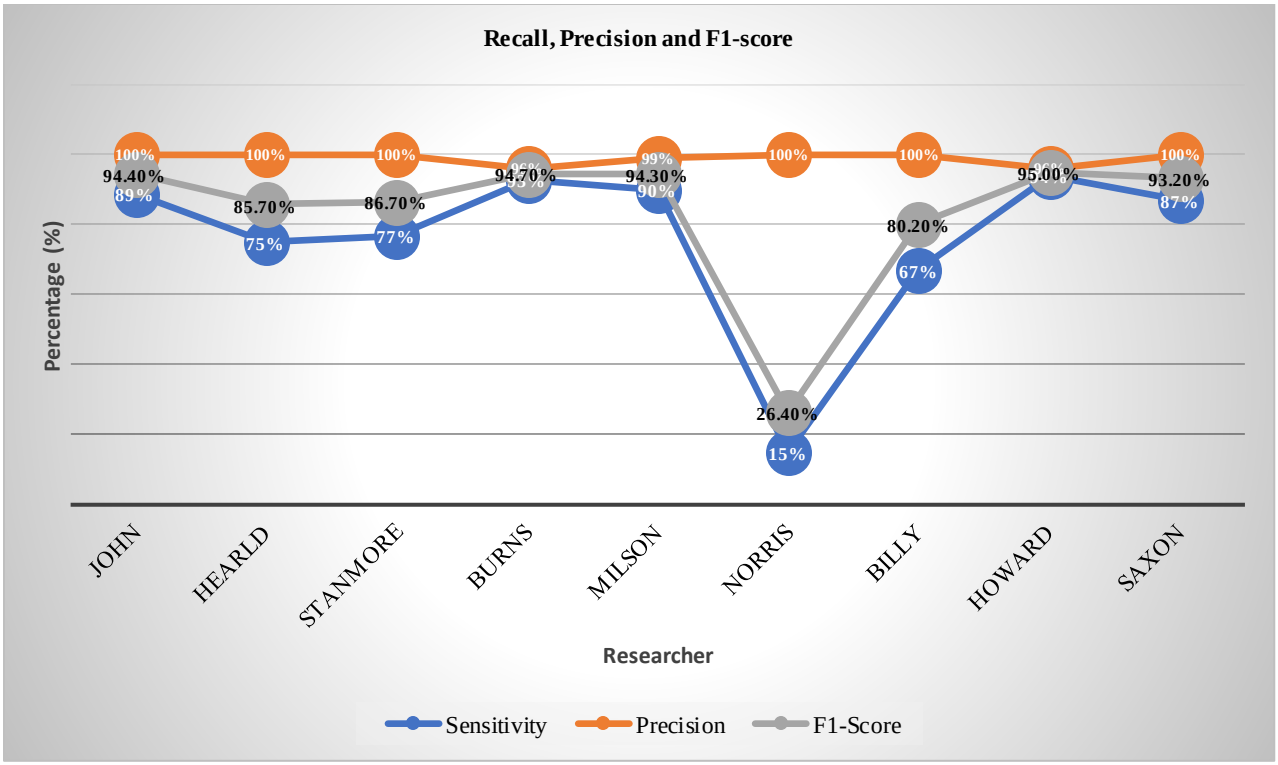

Figure 11 – Line graph of accuracy, sensitivity, specificity and precision metric comparisons for each sample researcher

Researcher	Sensitivity	Precision	F1-Score
JOHN	89%	100%	94.4%
HEARLD	75%	100%	85.7%
STANMORE	77%	100%	86.7%
BURNS	93%	96%	94.7%
MILSON	90%	99%	94.3%
NORRIS	15%	100%	26.4%
BILLY	67%	100%	80.2%
HOWARD	94%	96%	95.0%
SAXON	87%	100%	93.2%
Average	**76%**	**99%**	**85.7%**

Table 4 – Accuracy, sensitivity, specificity and precision metric comparisons for each sample researcher

As in *Figure 11* and *Table 4*, the sensitivity, precision and F1-score are used to evaluate the extraction and validation performance of our output 1 functionality. F1-score is the harmonic mean of both sensitivity and precision, providing a robust performance metric which both emphasises correctly extracted publications and penalizes wrongly extracted publications in evaluating our extraction model, balancing the metric properties of both sensitivity and precision. We can conclude that the overall performance of our method is satisfactory, with a high average F1-score of **85.7%** amongst our small sample**.** Interestingly, researcher *Norris* can be identified as an instance of significantly poor performance within our cohort, with a F1-score of only 26.4% and minimal

publications being extracted despite three of four of our extraction pipelines successfully being utilised– *please see the post-script for further analysis of this case.*

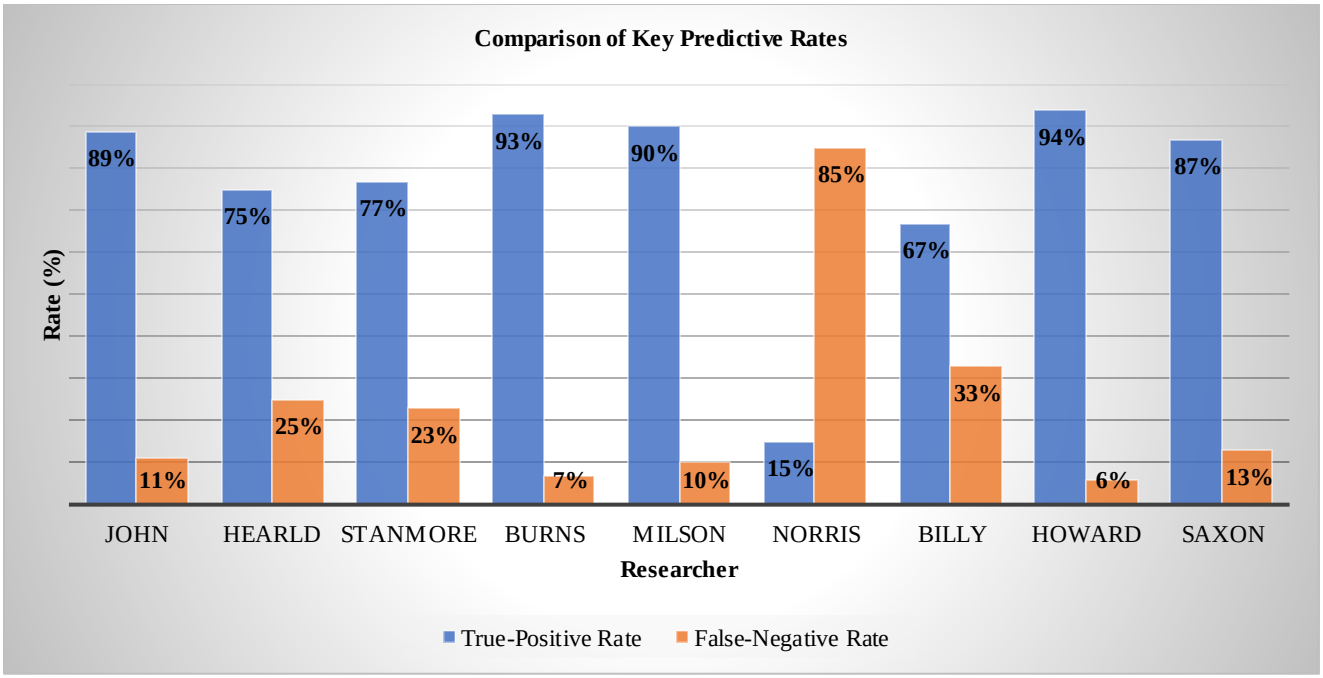

Figure 12 – Bar chart of comparisons between each sample researchers' predictive diagnostics rates

Researcher	True-Positive Rate	False-Negative Rate
JOHN	89%	11%
HEARLD	75%	25%
STANMORE	77%	23%
BURNS	93%	7%
MILSON	90%	10%
NORRIS	15%	85%
BILLY	67%	33%
HOWARD	94%	6%
SAXON	87%	13%
Average	**76%**	**24%**

Table 5 – Comparisons between each sample researchers' predictive diagnostics rates

Concluding our stage one to three evaluation, *Figure 12* and *Table 5* provide a comparison between the true positive rate and false positive rate of our extraction model, highlighting an average true positive rate of 76% and false positive rate of 24%. Overall, we can conclude that our extraction model is generally successful in extracting researchers publication profiles on our small researcher sample, with hyperparameter tuning providing room for improvement in reducing falsely extracted publications and significant enhancement in publication coverage likely obtainable through adjustment of the Scopus extraction pipeline – again, please *see the post-script for further discussion.*

5.2 Output 2: Researcher Expertise Representation

Output 2 is primarily concerned with stage four of our pipeline: research expertise representation and topic-modelling of researcher's publications. Below, *Figure 13* visualises our hyperparameter search for the optimal number of topics in hyperparameter tuning our LDA topic-model during our initial training phase, using *coherence* as our comparative metric for training over a large corpus of 10,000 cancer-related PubMed publications. With coherence generally decreasing with increasing topic numbers as opposed to perplexity, our maximum hyperparameter search range for finding an optimal topic number was set at below 100 topics. As we can observe, the highest coherence obtained searching globally over our arbitrary topic number range, and secondly in *Figure 13* then searching locally around the optimal global topic number of 10, returned a LDA model trained on 11 topics and with a coherence score of 0.46. This then suggests modelling 11 latent research topics within our training publication set to optimally capture the latent topic distributions within our cancer research literature sample, characterised by optimised human-interpretability according to the properties of coherence and within the limited context of comparisons based only on one language model type and implementation.

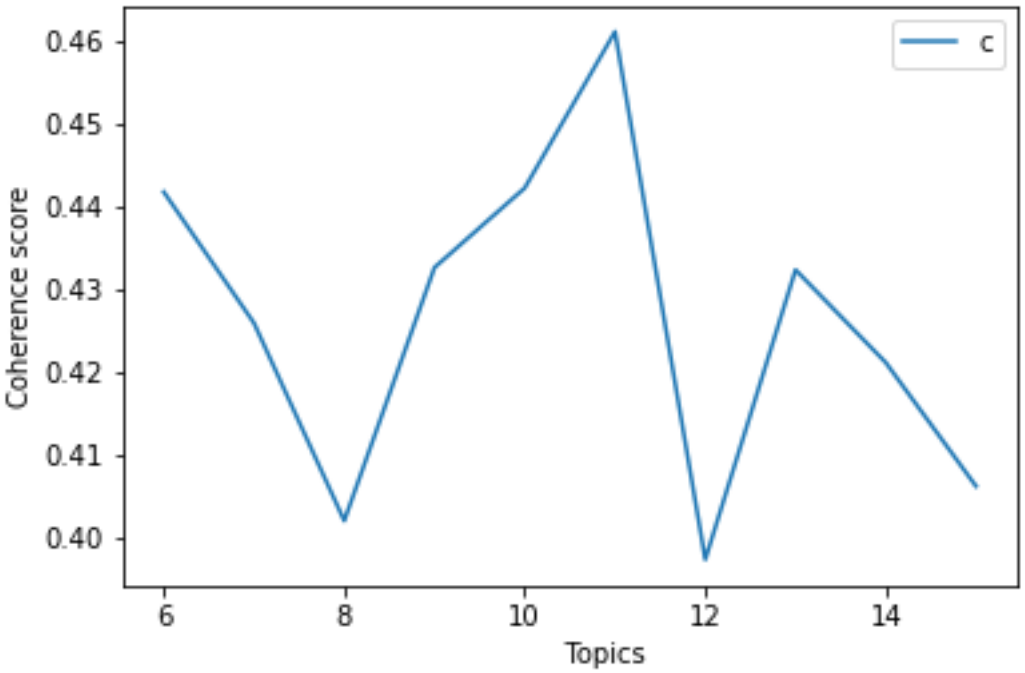

Figure 13 – Local hyperparameter search and comparison for training LDA topic model

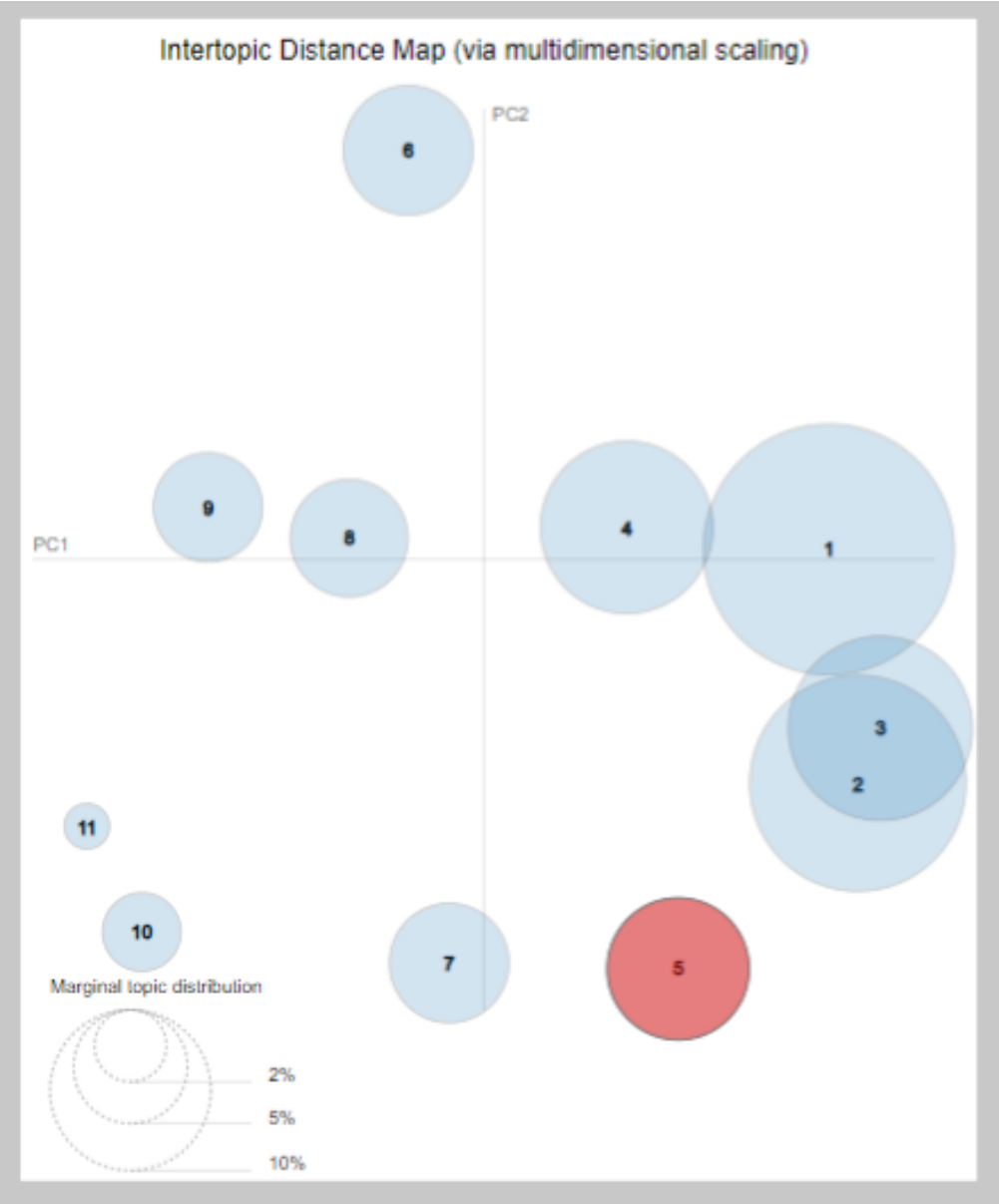

*Figure 14 – Interactive user-visualization of trained captured (cancer-related) topic distributions (*Sievert & Shirley, 2014)*

Following model training, using *PyLDviz's* LDA topic visualisation python package, users can interactively visualise the relative similarities and importance of each topic distribution modelled by our final trained LDA language model and corresponding identifying topic number. *Figure 14* provides a 'snap-shot' of this interactive topic-visualisation showing the inter-topic distance between modelled topic-word distributions of our researcher cohort through application of dimensionality reduction or 'multi-dimensional scaling', visualising topics through the first and second primary components of variation through Primary Component Analysis (PCA) (Sievert & Shirley, 2014). We can observe that topics 1 to 3 are closely associated, sharing topic-words over their topic-word probability distributions and further are identified as being three of the most important topics over the publication document corpus indicated by their large circle area.

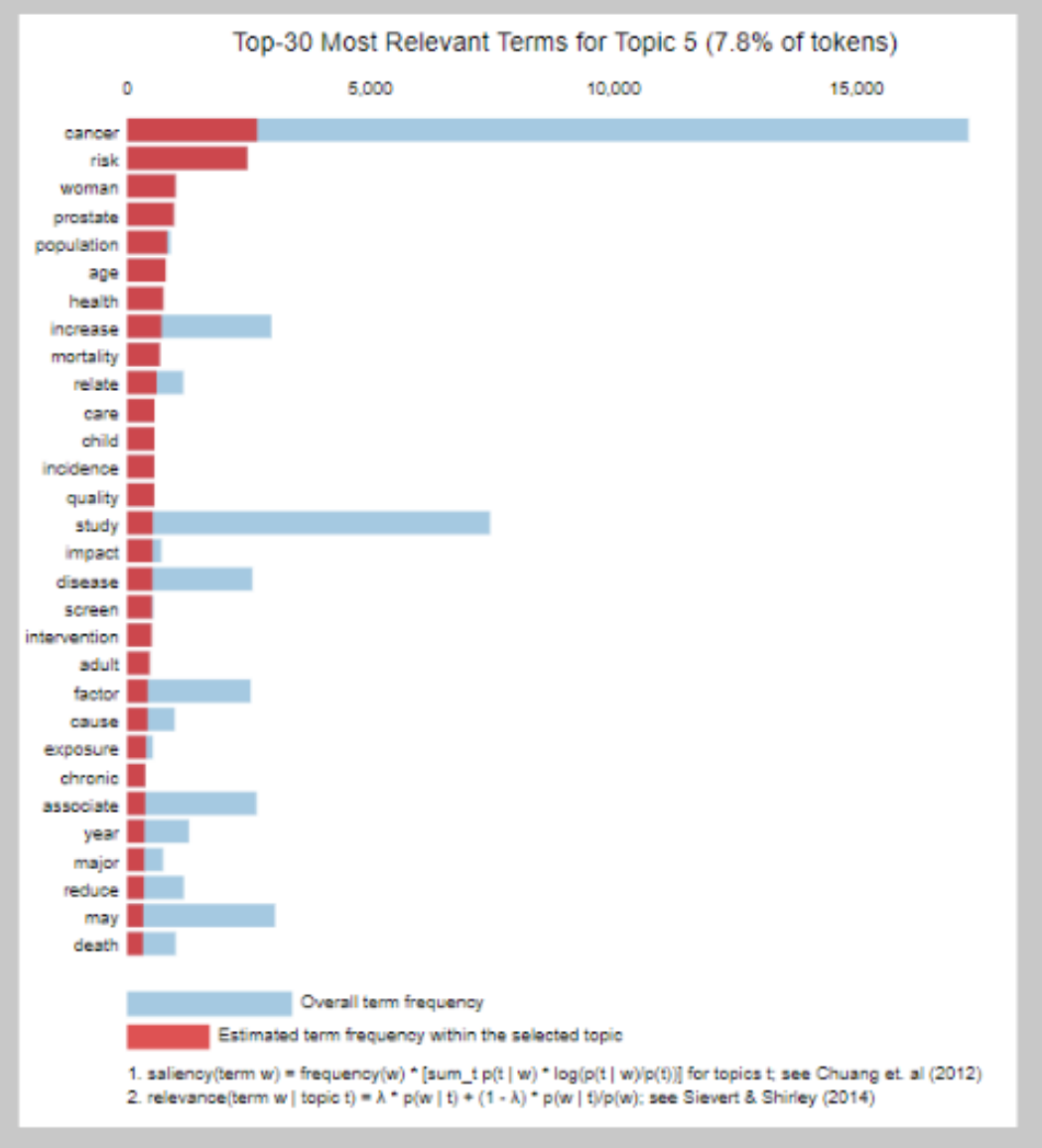

*Figure 15 – Example user-analysis and label allocation of a (cancer-related) topic word distribution by the research organisation user (*Sievert & Shirley, 2014)*

A key step in stage four requires the organisation-user to interactively analyse each returned topic distribution using PyLDviz's relevant topic word lists for each topic based on Sievert & Shirley's (2014) metric for topic-word relevance. Using this quantitative information and additionally qualitative understanding of their research organisations' specialities, users can then assign appropriate topic names for each topic distribution, thereby providing meaningful labels capturing each researcher's allocated research expertise. In the case of our research cohort analysis, the topic depicted in *Figure 15* was inferred to represent the '*Population Health Aspects of Cancer'* with high relevance words such as *'cancer', 'population', 'risk', 'disease', 'intervention', 'incidence' and 'mortality'* identified as being characteristic descriptors of population health as related to cancer research.

Topic Distribution Identity	User-Inferred Topic Label
1	Cancer Cell Biology
2	Clinical Epidemiology
3	Breast Cancer Research
4	T4
5	Population Health Aspects of Cancer
6	Chemotherapy
7	Genetic Aspect of Cancer
8	Cancer Diagnosis
9	T9
10	T10
11	Cancer Aetiology

Table 6 – Summary table of research topic labels and retained topic identifiers of UNSW research sample

Following visualisation and analysis over the 11 trained cancer-related research topics, *Table 6* summarises the topic labels allocated to each topic identifier where high interpretable confidence in the specific topic area was attained for each topic distribution, with our judgement and affiliation with the UNSW cancer research organisation standing in as the organisation-user. As illustrated, three topics (*T1*, *T9* and *T10*) were not able to be confidently named. However, our remaining 8 topics were identified as representing meaningful, human interpretable topics ranging from clinical, biomedical, genetic, cancer-specific (breast cancer) and population level aspects of cancer research.

Researcher	Cancer Cell Biology	Clinical Epidemiology	Breast Cancer Research	T4	Population Health Aspects of Cancer
JOHN	5.7%	18.3%	5.7%	11.4%	2.3%
HEARLD	5.4%	16.1%	6.2%	15.1%	2.3%
STANMORE	3.0%	8.1%	5.2%	9.8%	1.1%
BURNS	6.6%	17.0%	6.4%	14.3%	1.9%
MILSON	6.2%	12.3%	9.1%	15.6%	1.8%
NORRIS	4.6%	13.8%	6.0%	11.7%	3.8%
BILLY	7.5%	13.0%	5.7%	12.3%	2.4%
HOWARD	6.0%	16.0%	7.0%	12.0%	3.0%
SAXON	6.6%	14.9%	6.5%	12.8%	3.6%
Organisation Sample Average (%)	**5.7%**	**14.4%**	**6.4%**	**12.8%**	**2.5%**

Table 7 – Researcher topic spread proportions (%) (Topics 1 to 5)

Researcher	Chemotherapy	Genetic Aspect of Cancer	Cancer Diagnosis	T9	T10	Cancer Aetiology
JOHN	17.6%	13.7%	12.5%	7.4%	5.1%	0.2%
HEARLD	15.6%	14.3%	13.8%	6.6%	4.2%	0.4%
STANMORE	8.2%	10.1%	7.4%	4.1%	42.5%	0.3%
BURNS	14.3%	13.0%	12.8%	6.7%	6.1%	0.8%
MILSON	13.7%	15.1%	12.0%	7.1%	6.5%	0.5%
NORRIS	14.8%	12.5%	13.9%	11.3%	7.0%	0.5%
BILLY	15.0%	13.5%	12.7%	11.3%	6.0%	0.6%
HOWARD	13.0%	17.0%	12.0%	7.0%	7.0%	0.0%
SAXON	15.9%	13.3%	13.7%	7.5%	4.8%	0.5%
Organisation Sample Average (%)	**14.2%**	**13.6%**	**12.3%**	**7.7%**	**9.9%**	**0.4%**

Table 8 – Researcher topic spread proportions (%) (Topics 6 to 11)

Having trained and labelled topic distributions where appropriate, *Table 7* and *Table 8* present the topic profiling proportions of applying our trained LDA model to infer the topic distributions and percentages attributable to each of our primary researcher's 'unseen' publications.

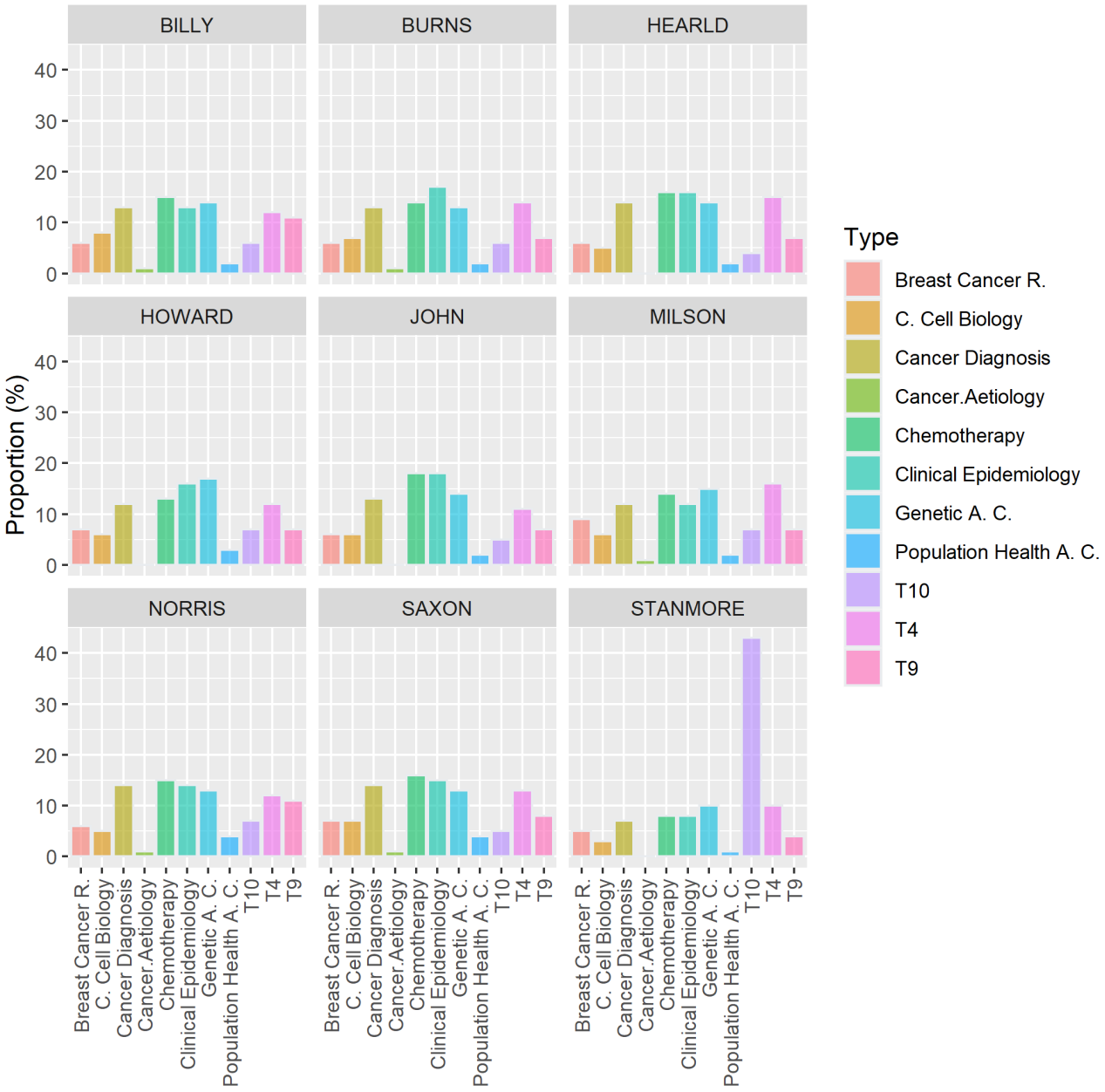

Figure 16 – Facetted histogram of research topic profiles for each sample researcher

Above, researchers' cumulative proportion of research topics spread over their set of publications are further visualised in *Figure 16* facetted by each researcher, with these tables and the above visualisation suggesting *Chemotherapy*, *Clinical Epidemiology* and *Genetic Aspects of Cancer* are three prominent cancer research topics amongst our cohort, with *Stanmore* being a particular outlier in the portion of research expertise allocated to the ambiguous topic 10. Overall, with our method illustrating the utility of capturing and allocating researcher topics using this approach, and potentially in providing research organisations with a map of their organisation's research expertise.

Researcher Expertise Representation Limitations

A key challenge and limitation in our current baseline topic modelling method results firstly from dependence on obtaining a training corpus of research publications large and representative enough to provide effective model training and learning of the diversity of researcher topic distributions amongst researchers, with optimisation of coherence for human interpretability of topic distributions largely dependent on training corpus size and diversity. With our current method using a predefined cancer publication corpus, our method assumes that researchers have not changed topic interests outside of the assumed cancer topics of the organisation. However, in reality, many researchers likely move between differing research specialities and interests throughout their research careers and thus their research expertise may not entirely be represented within a fixed organisation-based training corpus such as our '*cancer-related'* training corpus. To solve this limitation and provide a concrete functionality for stage four, additional components would need to be added to this pipeline in the future, incorporating a supplementary mechanism to automatically 'crawl' or through online bibliographic databases during extraction of primary researchers 'unseen' publications, extracting a proportionately and significantly larger training corpus not defined by the organisation's assumed specialities, rather being dependant and dynamically constructed from and 'around' primary researcher publication profiles through associated publication information such as paper keywords, iteratively being searched or "bootstrapped" during our researchers initial publication extraction in stage one.

5.3 Output 3: Research Collaboration Recommendations

i) Evaluation of Link Prediction/Research Collaboration Recommendations

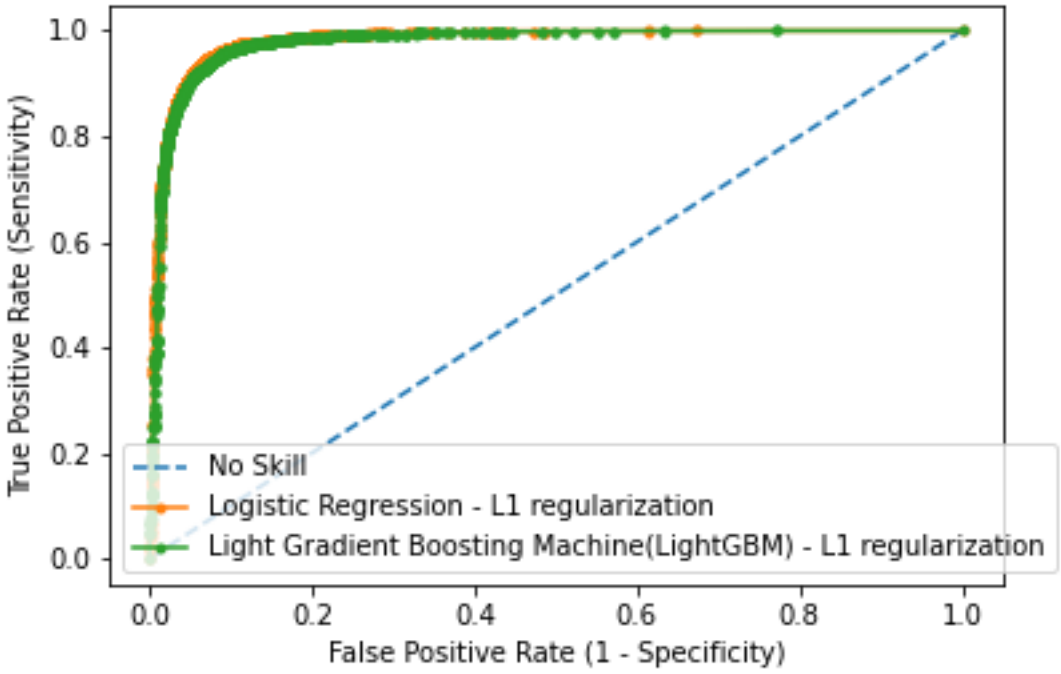

Figure 17 – Receiver operating characteristic (ROC) curve of two binary classifiers (LightGBM & Logistic Regression) and a 'no-skill' classifier as discrimination thresholds are varied

Following from stage five, Output 3 is concerned with the predictive performance of our heterogenous meta-path-based link prediction method described in our stage five methodology, comparing the performance of a trained simple logistic regression model against a more complex and highly-efficient gradient boosting decision tree model – Light Gradient Boosting (*LightGBM*) – on our static research collaboration graph network of topics, researchers, organisations and venues[conferences and journals]. LightGBM is an implementation of the Gradient Boosting Decision Tree (GBDT) algorithm designed to reduce the computational cost of popular GBDT implementations such as XGBoost, combining *Exclusive Feature Bundling* (EFB) and *Gradient-based One-Side Sampling* (GOSS) as an algorithmic solution to accelerate the training process of GBDT's whilst achieving near the same levels of accuracy (Ke et al, 2017). We chose to compare these two models as a reflection of complexity against simplicity and in comparing applications of a traditional vs. more sophisticated model in providing research collaboration recommendations and link predictions.

Our approach found that during training and validation models trained on researcher link embeddings between two researcher embedded nodes using a L1 regularisation link operator, rather than L2 regularisation link operator, resulted in better performing models for our research cohort in terms of Area Under the Curve (AUC) performance. Thus, with our method automatically using L1 regularisation as a baseline for link embedding, it appears the performance of both the Logistic Regression and LightGBM was similar, however, with the baseline logistic regression model slightly outperforming the more complex LightGBM. This suggests that predicting

whether two researchers who have not previously collaborated as co-authors on a publication, will in future collaborate and form a co-authorship edge is viable using our heterogenous network pipeline and a traditional generalised linear model on a sampled static heterogenous graph network and with a small cohort of 9 primary researchers and 2665 secondary researchers, with both logistic regression and LightGBM achieving high AUC scores of 98% and 97.8% respectively.

Link Prediction Pipeline Limitations

A significant limitation in our link prediction method is characterised by the dependence of predictions based on randomly sampled edges from a static graph network representation, with an alternative method based on a temporal research graph network or dynamic graph, with time represented by publication year and splitting of training and testing datasets by yearly periods, which is likely to more robustly allow for evaluation of predictive performance based on realistic predictions over successive periods of time rather than dependence on random sampling of a static graph. We therefore recommend a temporal method, and evaluation of this is likely to be necessary to gain insight into the practical performance of these recommendations in the development context of successfully predicting and recommending useful research collaborations within the research environment overtime (Sinha, Cazabet & Vaudaine, 2018). Unfortunately, time did not permit the exploration of temporality in these prediction models.

ii) Example Analysis of Two Research Collaboration Recommendations for a single primary researcher

Having automatically trained and chosen our final logistic regression model which performed best at predicting 'new' or sampled co-author research collaboration edges between researcher pairs from our static heterogenous graph, this final model can then be used to predict or provide research collaboration recommendations for researchers, using as input the entire remaining research graph network with all current co-authorship edges. Furthermore, such a dataset represents all co-authorship collaborations between primary researchers and secondary researchers up to the current date.

The table below illustrates the result of these predictions, being computed by using our final logistic regression model on the entire research collaboration static graph, without adherence to a development dataset and thus serving only as proof-of-concept example of the output and utility of research collaboration recommendations on a non-temporal static graph.

Recommendation Type:	Secondary Recommendation			
Primary Researcher:	BILLY, M			
Secondary Researcher:	JAMES, C			
Research Collaboration Probability:	53.24%			
Intersection	**Entity**	**Researcher1-Entity-Weight**	**Researcher2-Entity-Weight**	**Entity-Network-Weight**
Topic	Population Health Aspects of Cancer	6.63	0.25	1992
Topic	Cancer Cell Biology	3.00	0.28	1964
Venue	BMI open	5	1	632
Co-author	PHI, P	2	4	168

Table 9: Recommendation direct-relationship inference summary table for BILLY, M & JAMES, C

From *Table 9* we can observe, primary researcher *Billy, M* and *James, C* – a secondary researcher – according to our model have a 53.24% probability (that is, one-in-two chance) of collaborating in the publication of a paper, having never been co-authors together in the past. We suggest that researchers such as these who have moderate probability of collaborating are those that serve to gain the most in active fostering of research collaborations, as those pairs with significantly higher probability of publication are assumed to require less or minimal active support by research organisations due to their likely existing proximity within a research organisations network. Thus, suggesting that researchers such as these with recommendation probabilities higher than random chance may be of interest to research organisations for supporting research collaborations in the organisation. *Table 9* and *Table 10* aim to infer degrees of direct similarity for each recommendation, displaying direct relationships or intersections in shared node relations. We can observe our two researchers from *Table 9* both share expertise in *Population Health Aspects of Cancer* and *Cancer Cell Biology*, with these being a significantly important entity in terms of edges in our overall research network, with 1992 and 1964 mixed type edges within our heterogenous research network overall. Further, these researchers have both published in *BMI Open* Journal and have been co-authored with a significant secondary researcher in the network, *Phi, P*.

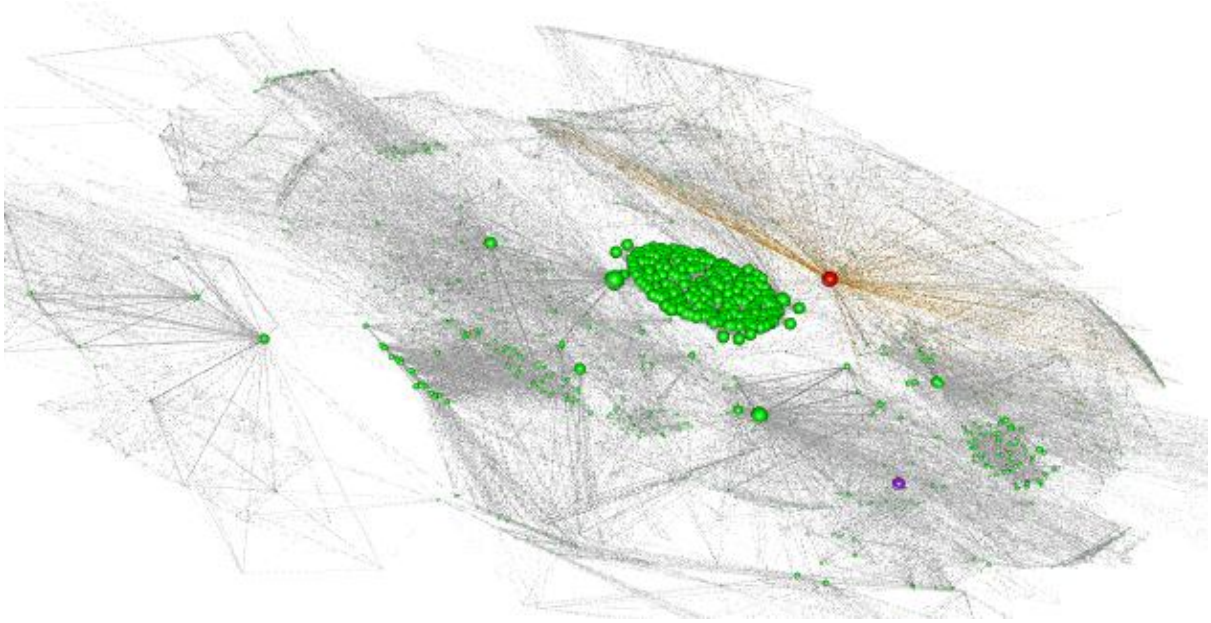

Figure 18 – Recommendation indirect-relationship inference using co-author network visualisation for Billy, M & James, C

Additionally, *Figure 18* visualises the co-authorship network, highlighting our two researchers involved in the above secondary recommendation, with an orange node indicating our primary researcher and purple node indicating our secondary researcher. We can observe indirectly the large distance between these researchers in terms of co-authorship relations, thereby confirming the need for existing relationships beyond just shared co-authors as a basis for similarity or proximity within our heterogenous research network and this recommendation; evidenced in *Table 9* with numerous direct non-co-author intersections existing between these two researchers.

Recommendation Type:	Secondary Recommendation			
Primary Researcher:	BILLY, M			
Secondary Researcher:	RILEY, T			
Research Collaboration Probability:	50.43%			
Intersection	**Entity**	**Researcher1-Entity-Weight**	**Researcher2-Entity-Weight**	**Entity-Network-Weight**
Co-author	LUCAS, D	2	4	291
Co-author	AZZOPARDI, C	2	2	287
Co-author	EGBERS, E	2	4	291
Co-author	VORLEY, A	2	8	311
Co-author	GREEN, A	2	2	288
Co-author	JONES, A	2	2	283

Table 10: Recommendation direct-relationship inference summary table for RILEY, T & BILLY, M

As another suggested research collaboration recommendation, we again observe *Billy, M* is involved in another secondary recommendation, having a 50.43% chance of collaborating with previously unassociated secondary researcher *Riley, T*. Unlike our previous recommendation all intersections for these two researchers is based on shared co-authors, with our two researchers evidently closer in distance within the below co-authorship network of *Figure 19*.

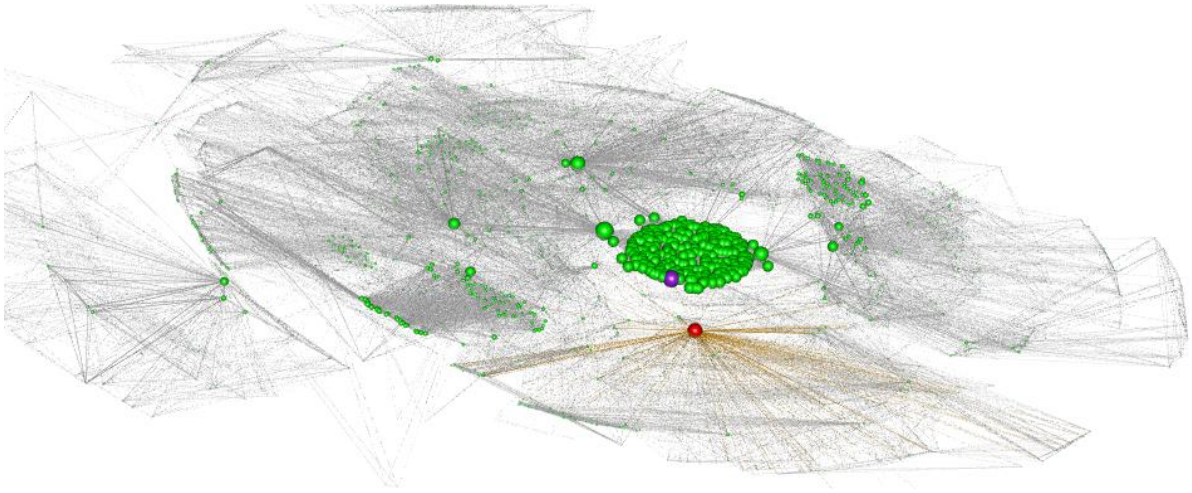

Figure 19 –Recommendation indirect-relationship inference using co-author network visualisation for RILEY, J & BILLY, M

5.4 Output 4: Research Network Visualization

i) Top-10 most influential entities (Researchers, Venues, Organisations, Research Topics) of the Heterogeneous Research Collaboration Network

For the final stage of our pipeline, output 4 provides in-depth research collaboration network visualisations from two research collaboration network perspectives. Firstly from a simple homogeneous co-authorship network perspective, all primary and secondary researchers in stage five and their position, research recommendation involvement and importance are mapped and visualised for all primary researchers and key secondary researchers within the context of the organisations co-authorship network. Furthermore, from a second heterogeneous or complex network perspective, the heterogeneous mixed-type or multi-layer network construction used in stage five of research topics, researchers, organisations and venues, is displayed to users providing a wide in-depth perspective of the research collaboration ecosystem of the organisation, through both heterogeneous multilevel network visualisation and secondly through network statistics tables of each key entity type. Ultimately, with our final multilevel visualisation providing insight into the most significant and influential research topics, venues, organisations and researchers together in the network, their extent of connectivity through node density, and finally key relationships of the organisation between differing researcher entities of the research organisations collaboration network.

Venue	Unnormalized Density
International Journal of Radiation Oncology Biology Physics	619
Journal of Clinical Oncology	543
Journal of Palliative Medicine	528
BMJ Open	361
Journal of Pain and Symptom Management	328
Medical Journal of Australia	306
Paediatric Pulmonology	266
Journal of Adolescent and Young Adult Oncology	255
Lancet Oncology	250
Asia-Pacific Journal of Clinical Oncology	243

Table 11 – Top-10 most influential venues based on node degree

Organisation	Unnormalized Density
University of New South Wales	443
University of Sydney	116
Sydney Children's Hospital	106
Western Sydney University	105
Liverpool Hospital	85
University of Technology Sydney	71
Ingham Institute for Applied Medical Research	71
Prince of Wales Hospital	50
Flinders University	47
University of Queensland	42

Table 12 – Top-10 most influential organisations based on node degree

Researcher	Unnormalized Density
MILSON, D	675
JOHN, S	523
HOWARD, B	481
BILLY, M	475
STANMORE, Q	351
MERTIN, J	304
MCKEE, R	293
FAYAS, P	288
CLARK, J	288
JAME5, D	288

Table 13 – Top-10 most influential researchers based on node degree

Topic	Unnormalized Density
Cancer Cell Biology	523
Population Health Aspects of Cancer	509
Breast Cancer Research	508
Chemotherapy	501
Genetic Aspects of Cancer	499
Clinical Epidemiology	481
Cancer Diagnosis	480
T9	469
T4	393
T10	139
Cancer Aetiology	0

Table 14 Topics ordered by highest network influence (node degree)

Table 11, *Table 12*, *Table 13*, and *Table 14* above display the top-10 most influential entity nodes for each entity type in our cohorts' network, thus providing an understanding of the organisations most influential research entities within their network. In *Table 11* we can infer the three most influential research journals for our research cohort is the international *Journal of Radiation Oncology Biology Physics*, *Journal of Clinical Oncology* and *Journal of Palliative Medicine*, with these being respective top-ranking outliers with node degree or level of connectivity of 619, 543 and 528 edges respectively, as compared with other subsequently lower ranked venue entities in the network ranging around 300 heterogeneous edges. Meanwhile, *Table 14* provides insight into the most prominent research topics for the network as topic-modelled in stage four, suggesting *cancer cell biology*, *population health aspects of cancer*, *breast cancer research*, *chemotherapy genetic*, *clinical epidemiological aspects* and *cancer diagnosis* as being all significant within the research network. In terms of organisations, we can see *University of New South Wales* which is our cohorts primary organisation, is most prominent, with the *University of Sydney* being a major university research partner for our cohort, followed by the *Sydney Children's Hospital* and *Western Sydney University*. Finally, in terms of the most influential researchers both primary and secondary within the organisations research network, we can observe 5 of our 9 primary researchers are amongst the top 5 ranked most influential researchers of the network, with numerous secondary researchers then presented as important research collaborators likely affiliated with our primary organisations most prominent partnered research organisations.

ii) Homogenous Co-authorship Network Visualizations using *MuxViz* & *Gephi Visualization Software*:

Figure 20 and *Figure 21* below provide two snapshots from differing angles of an interactive visualisation session in R, leveraging MuxViz as an Rshiny application to provide users with an overview of their organisation's homogenous co-authorship network. Primary researchers are viewed as orange or red nodes in the network, with orange signifying those primary researchers not involved in a research collaboration recommendation from stage five, and red signifying researchers involved within a research collaboration recommendation. Furthermore, purple researcher nodes indicate secondary researchers involved in a research collaboration with a primary researcher, whilst green nodes mark general secondary researchers of the organisations network. Additionally, white secondary co-authorship edges and orange primary research edges

provide the relational distance and community structure within our research network, with node size then indicating degree or importance of each researcher within the network. We can observe overall, there are three primary researchers (red) and two secondary researchers (purple) involved within secondary recommendations, with two of these purple secondary researchers and one of the red primary researchers thus being our primary researcher *Billy* and two secondary researchers (*James* and *Riley*) from our output 3 recommendation results.

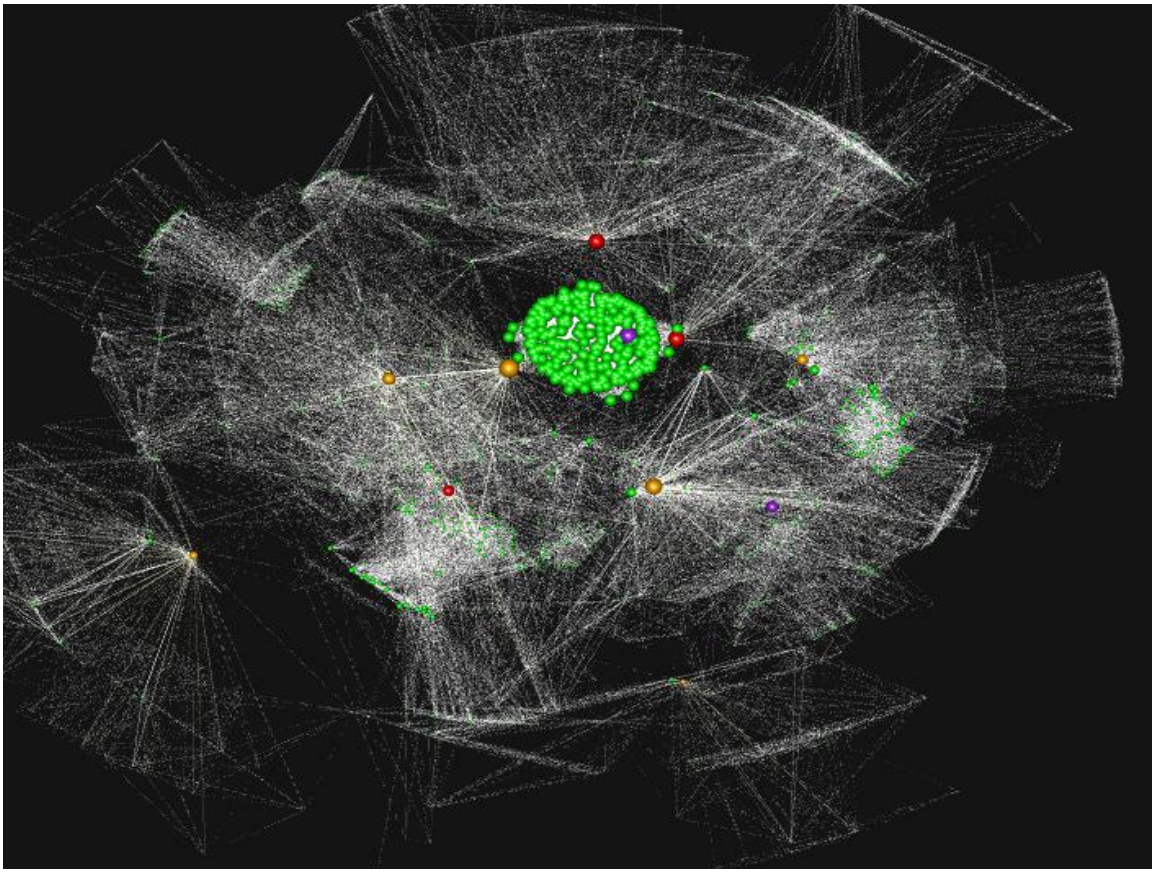

Figure 20 – Co-authorship network visualisation angled top-down view (MuxViz)

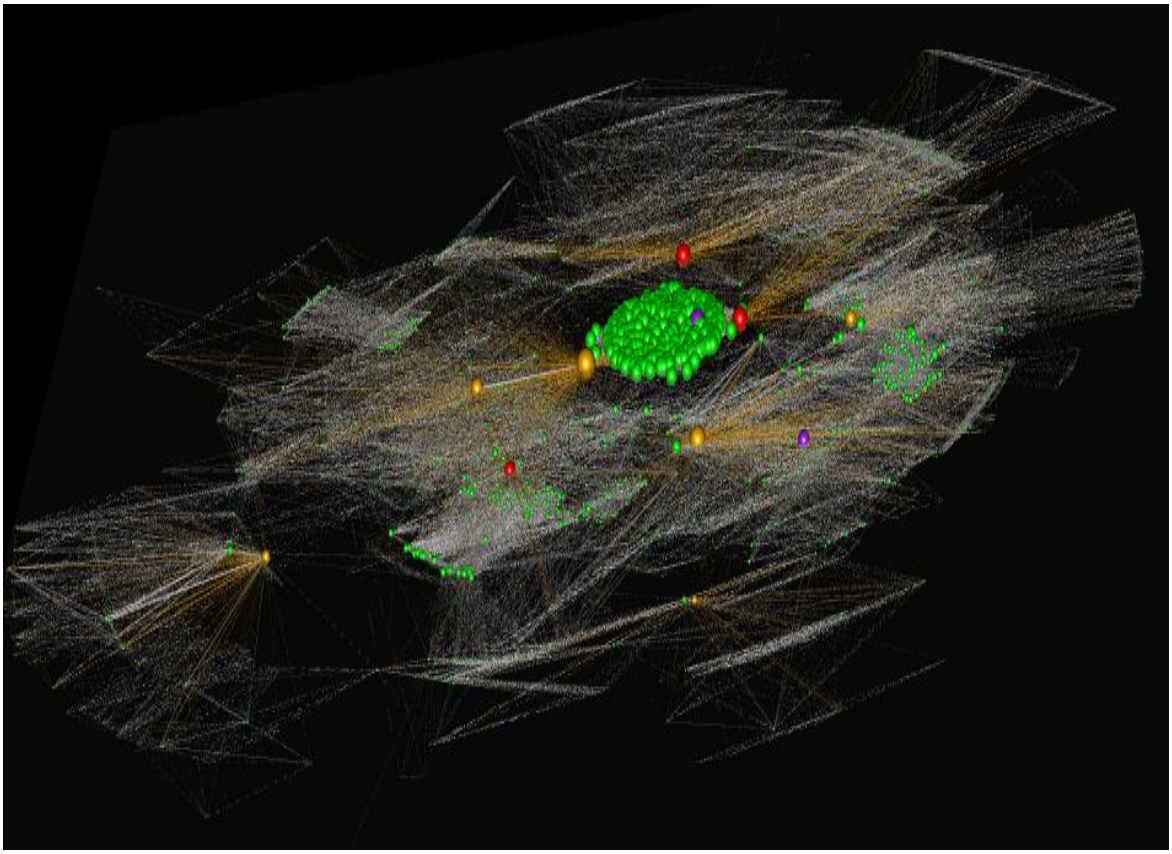

Figure 21 – Co-authorship network visualisation; angled-landscape view (MuxViz)

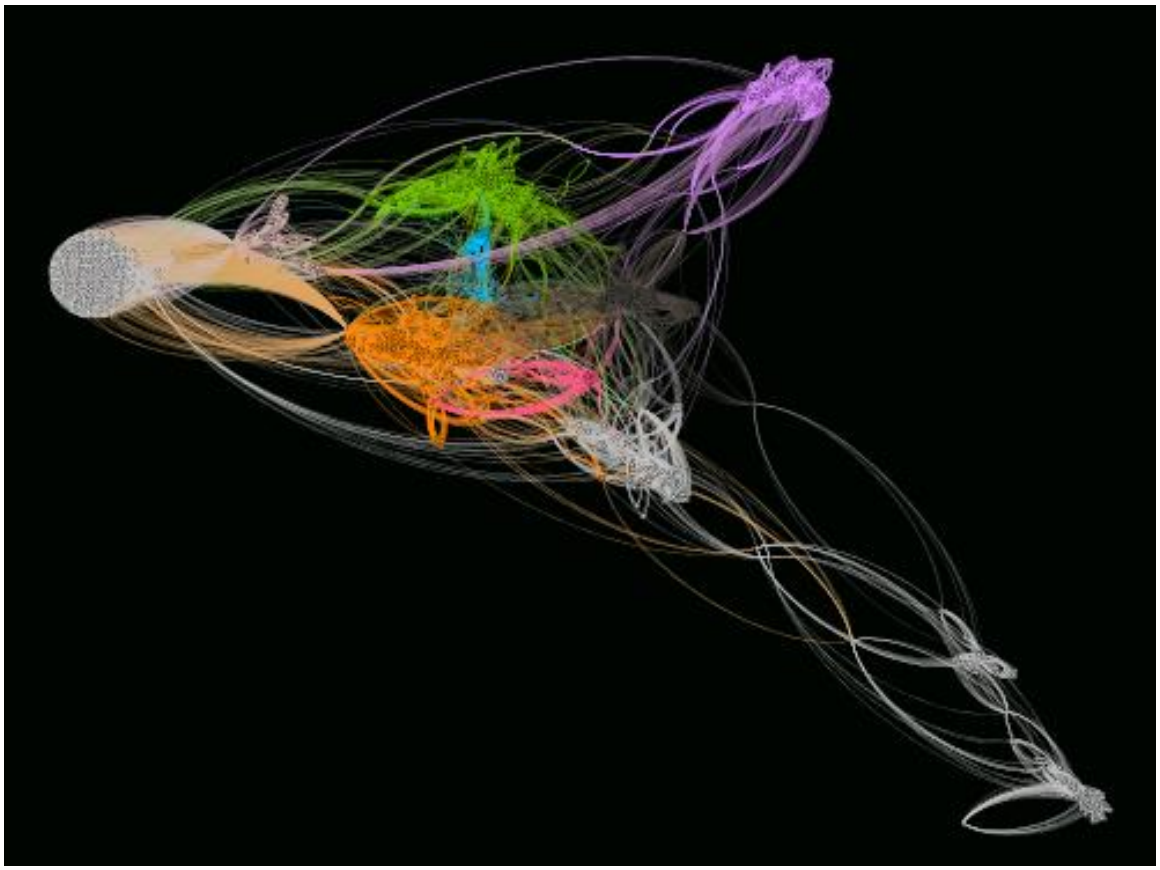

Figure 22 – Co-authorship network visualisation with community modularity (Gephi Visualization Software)

Using the *Gephi Visualization Software* for network visualisation, the above output of stage six allows users to import *Gephi* files created within our stage six pipeline to explore their co-authorship network interactively with an array of network analysis algorithms and custom visualisation tools leveraging the Gephi visualisation dashboard. *Figure 22* displays a Gephi Visualization of the research organizations co-authorship network, applying Gephi's community modularity algorithm to detect shared underlying community structure within the organisations research collaboration network; measuring how vertices in each community share edges as compared with an otherwise randomised network (Mohadeseh et al, 2015). Interestingly, we can observe there are eight distinct communities discovered, with a significantly large white community spanning along different researcher clusters within the network.

iii) Heterogeneous Interdependent Multilevel Research Collaboration Network Visualizations using MuxViz

Finally, the third and final output of stage six provides the most in-depth research network visualisations of our tool, utilising 2-dimensional and 3-dimensional functionality in *MuxViz* to deliver interactive exploration of the complex heterogeneous research network and each corresponding network layer, filtering this network using node degree of research entities to capture and display only the most influential entities and relationships characterised by the research organisation through the lens of researchers, modelled research topics, organisations and published venues. Below *Figures 23-28* provide differing angled-snapshots of interactive 360-degree visualisation sessions in both 2-D and 3-D formats using *MuxVi*x based on our research cohort. We can

observe amongst our nine primary researchers (orange and purple nodes), that there are three prominent primary researchers that share collaborative relationships as a researcher cluster, with other primary researchers distanced as outliers to this collaborative cluster. Furthermore, the *University of Technology Sydney* and *Flinders University* are visualised as a significant partnered organisation for this cluster and further *BMJ Open*, *Journal of Clinical Oncology* and the *Journal of Palliative Medicine* are identified as venues in which these researchers often publish. At the level of organisations, there appear to be three prominent collaborative organisation communities amongst key organisations of the research network, with our primary organisation, the *University of New South Wales*, closely collaborating with the *Ingham Institute for Applied Medical Research* and *Western Sydney University*. Furthermore, looking at the *University of New South Wales* we can observe that there are strong expertise relationships with *Cancer Cell Biology* and *Population Health Aspects of Cancer* amongst publications of associated primary researchers represented by this UNSW cohort. Finally, through the lens of topic expertise we can conclude that a majority of research expertise is clustered around 6 topics, with *Clinical Epidemiology* (though an important topic in terms of degree or influence in the network) appearing to be an outlining topic with regard to sharing similar researchers who published publications characterised by this research interest covering additional research topics.

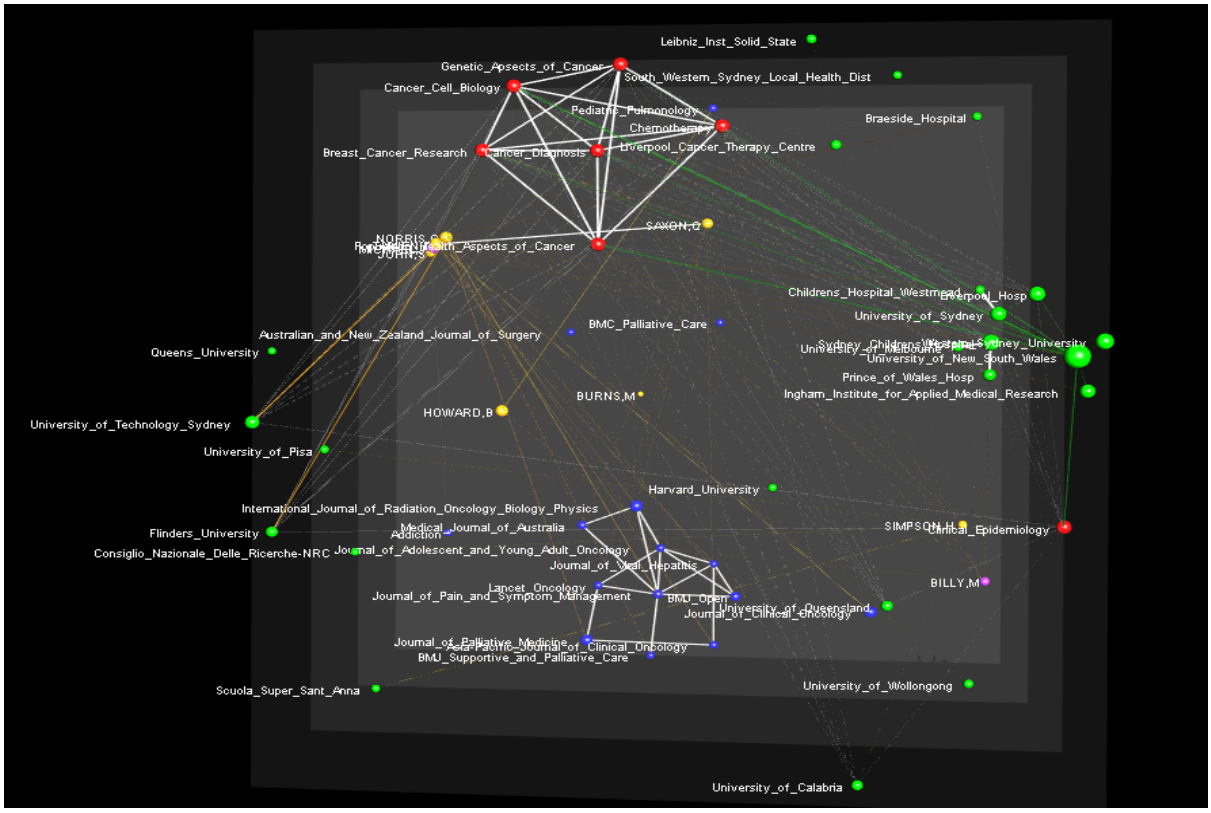

Figure 23 – Heterogeneous multi-level research collaboration network, aggregated front-panel view (2-D)

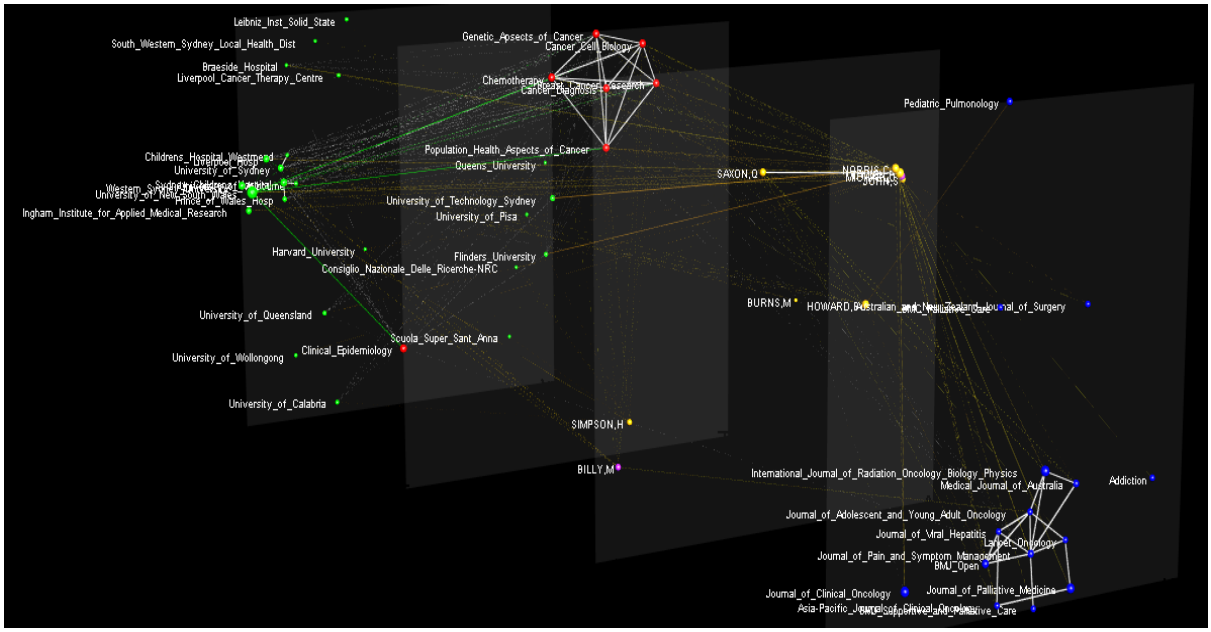

Figure 24 – Heterogeneous 2-D multi-level research collaboration network, venue-layer angled view (2-D)

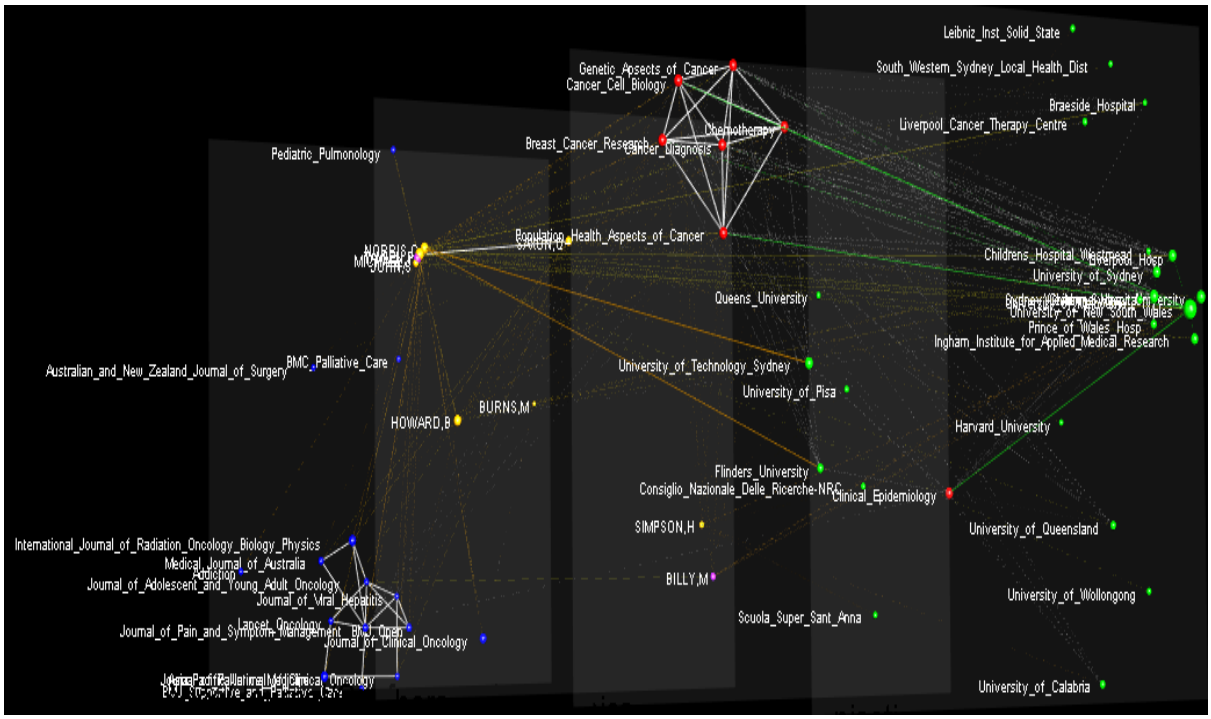

Figure 25 – Heterogeneous multi-level research collaboration network, organisation-layer angled view (2-D)

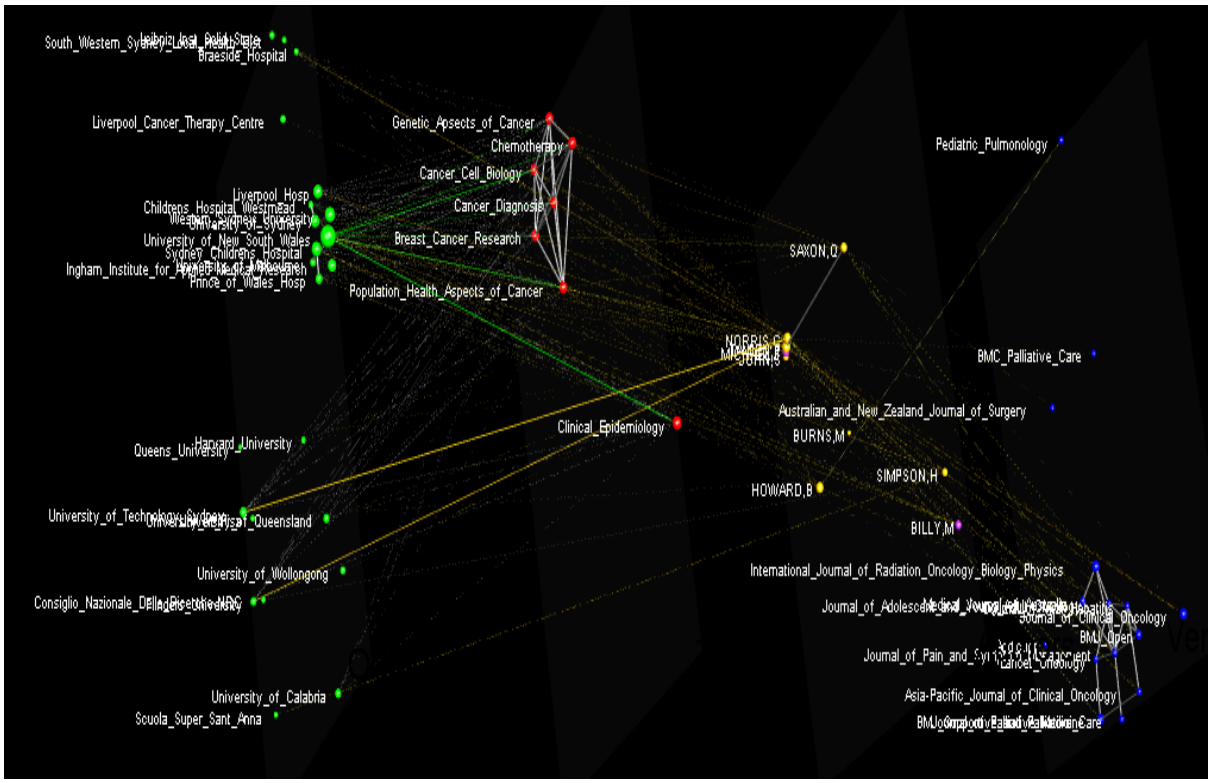

Figure 26 – Heterogeneous multi-level research collaboration network, venue-layer angled (3-D)

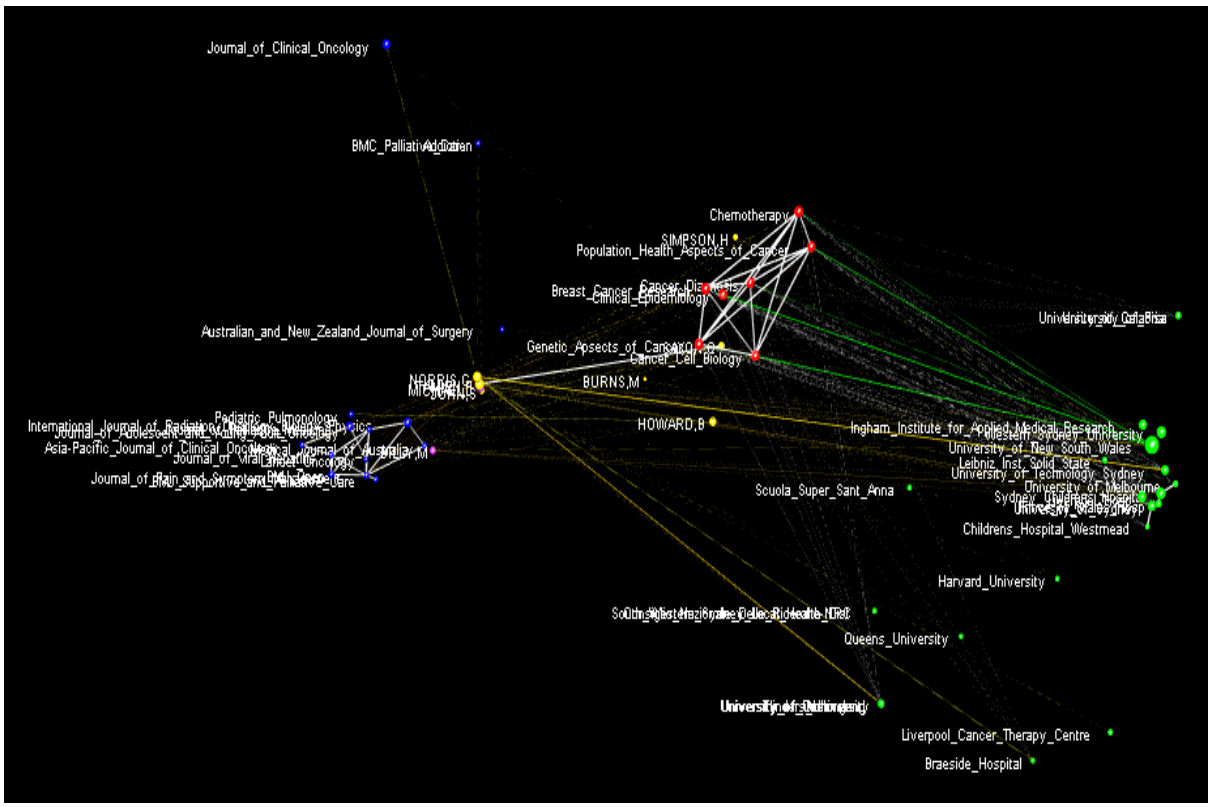

Figure 27 – Heterogeneous Multi-level Research Collaboration Network, organisation-layer angled (3-D)

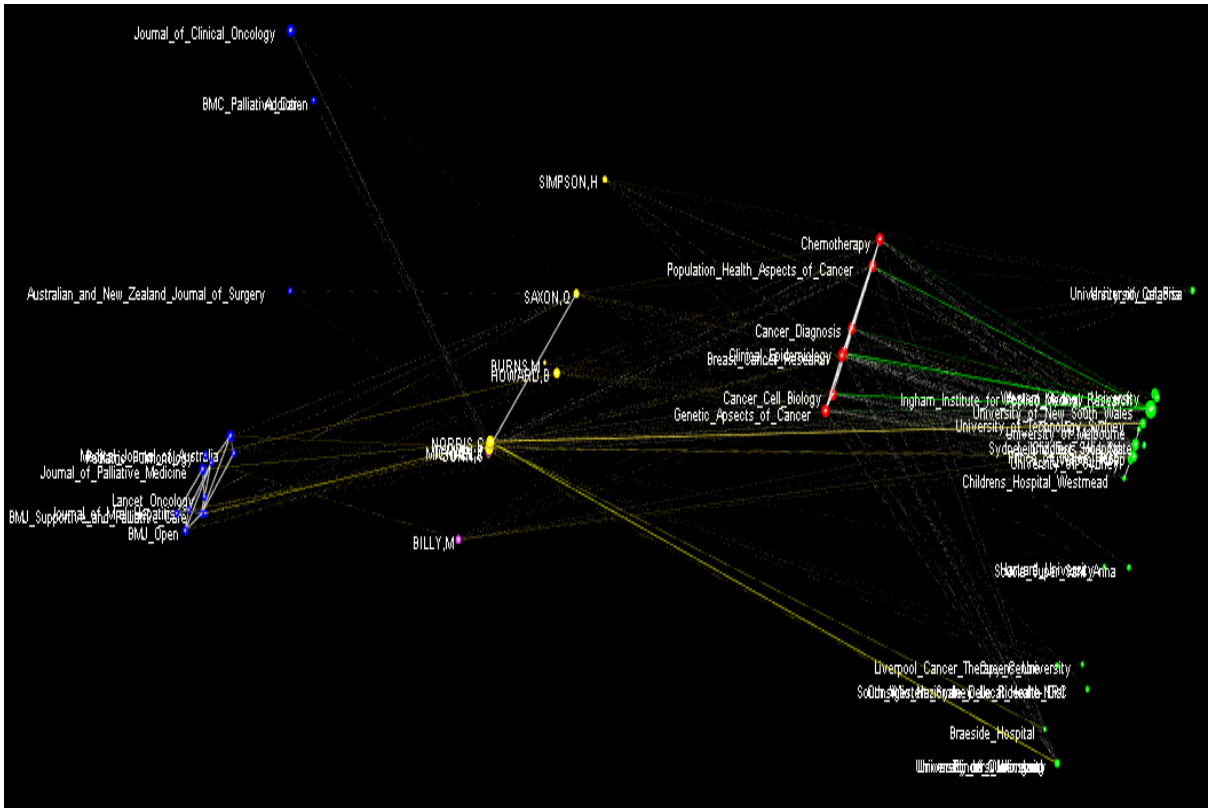

Figure 28 – Heterogeneous 3-D Multi-level Research Collaboration Network, side-view (3-D)

6.0 Conclusion

We believe that our proof-of-concept toolkit illustrates the opportunity that currently exists for research organisations to utilise the increasingly abundant and accessible sources of research data available on their researchers and research outputs, being actively collected through the many online bibliographic repositories that freely offer their services to the research community. Combined with recent advances in machine learning and data visualisation technologies, these untapped sources of research organisation intelligence can be used to serve an increasingly active and iterative role within these organisations: that of recommending or otherwise encouraging collaboration. Tools such as our proof-of-concept toolkit thus have potential to serve as a foundation for research organisational insight and increased support and integration of research departments and centres within research organisations, establishing pipelines from online bibliographic data sources to research organisations databases, applying extraction methods and disambiguation techniques showcased within our toolkit to iteratively update and model researcher publication profiles with researcher expertise and learn and visualise collaborative patterns of the past between research departments, centres and partnered organisations to leverage research big data in providing valuable research collaboration recommendations for supporting, strengthening and actively building research collaboration networks into the future. Furthermore, with fostering of these collaborations between researchers that should, or could, exist but do not is important because all indicators point to “big science” and large collaborations being more successful and more efficient than small teams.

Serving as a proof-of-concept, our tool thereby highlights the future utility and practicality of applying pipelines of these methods to freely available research data. Although only a very small proof-of-concept initial dataset was used in the work reported here – just nine researchers – the toolkit mostly performs well on both researcher publication profile extraction and disambiguation, in predicting static research collaborations within a research network for research collaboration recommendations, modelling researcher expertise and delivering useful network visualisations for research organisations. Larger input datasets, such as the several hundred cancer researchers active across all of UNSW, is expected to yield even better results and further insights.

In conclusion, our proof-of-concept toolkit provides a framework and pathway for the delivery of more efficient and user-friendly applications of this kind in the future; ultimately enabling research organisations to make better use of research output data beyond simple per-researcher comparisons that dominate academic performance management currently.

7.0 References

Afzal, M. and Maurer, H. (2011). Expertise Recommender System for Scientific Community. *Journal of Universal Computer Science*, 17, 11, 1529-1549.

Aisheh, Z., Raveaux, R., Ramel, J., & Martineau, P. (2015). An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems. *4th International Conference on Pattern Recognition Applications and Methods*. doi: 10.5220/0005209202710278.

Allaire, Ushey, J., Tang, Y., and Eddelbuettel, D. (2017). reticulate: R Interface to Python. *RStudio*. https://github.com/rstudio/reticulate

Backes, T. (2018). The Impact of Name-Matching and Blocking on Author Disambiguation. CIKM'18: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, 803-812. doi: 10.1145/3269206.3271699.

Bastian, M., Heymann, S,. & Jacomy, M. (2009). Gephi: an open source software for exploring and manipulating networks. In: *Proceedings of the Third International Conference on Weblogs and Social Media*.

Blei, D., Ng, A., & Jordan, M. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3, 993-1022.

Bozeman, B., Fay, D., & Slade, C. (2013). Research collaboration in universities and academic entrepreneurship: the-state-of-the-art. *The Journal of Technology Transfer*, 36, 1-67.

Campello, R., Moulavi, D., & Sander, J. (2013). Density-Based Clustering Based on Hierarchical Density Estimates. In *PAKDD*.

CSIRO's Data61. (2018). StellarGraph Machine Learning Library. *Github Repository, Github*. https://github.com/stellargraph/stellargraph.

De Domenico, M., Porter, M., & Arenas, A. (2014). MuxViz: a tool for multilayer analysis and visualization of networks. *Journal Of Complex Networks*, *3*(2), 159-176. doi: 10.1093/comnet/cnu038

Fize, J., Oliver, C., & Hoksza, D. (2018). Ged4Py. *GitHub Repository, Github*. https://github.com/Jacobe2169/ged4py/commit/8cd3918df43eb5a36d283a9f8674f3b145054ab7

Frey, B., & Dueck, D. (2007). Clustering by Passing Messages Between Data Points. *Science*, 315, 5814, 972-976.

Fuccella, V., De Stefano, D., Vitale, M., & Zaccarin, S. (2016). Improving co-authorship network structures by combining multiple data sources: evidence from Italian academic statisticians. *Scientometrics*, 107, 167-184. doi:10.1007/s11192-016-1872-y

Haruna, K., Akmar Ismail, M., Damiasih, D., Sutopo, J., & Herawan, T. (2017). A collaborative approach for research paper recommender system. *PLOS ONE, 12*, 10, e0184516. doi: 10.1371/journal.pone.0184516

Katsurai, M., Ohmukai, I. and Takeda, H. (2016). Topic Representations of Researchers' Interests in a Large-Scale Academic Database and Its Application to Author Disambiguation. *IEICE Transactions on Information and Systems*. 9,4, 1010-1018.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *31st Conference on Neural Information Processing Systems*.

Li, J., Zhao, D., Ge, B., Yang, K., & Chen, Y. (2018). A link prediction method for heterogeneous networks based on BP neural network. *Physica A*, 495, 1-17.

Liben-Nowell, D. & Kleinberg, J. (2007). The Link-Prediction Problem for Social Networks. *Journal for the American Society for Information Science and Technology*, 58, 7, 1019-1031.

Mimno, D., Wallach, H., Leenders, E., & McCallum, A. (2011). Optimizing Semantic Coherence in Topic Models**.** *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing* , 262–272.

Mohadeseh, G., Abbas, S., Hosein, A., Bailey, J., & Stuckey, P. (2015). Generalized Modularity for Community Detection. Joint European Conference on Machine Learning and Knowledge Discovery in Databases: Machine Learning and Knowledge Discovery in Databases, 655-670.

Momtazi, S., & Naumann, F. (2013), Topic modeling for expert finding using latent Dirichlet allocation. *WIREs Data Mining Knowl Discov*, 3, 346-353. doi:10.1002/widm.1102

Parikh, R., Mathai, A., Parikh, S., Chandra Sekhar, G., & Thomas, R. (2008). Understanding and using sensitivity, specificity and predictive values. *Indian Journal of Ophthalmology*, *56*, 1, 45. doi: 10.4103/0301-4738.37595

R Core Team. (2017). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

Rode, H., Serdyukov, P., Hiemstra, D., & Zaragoza, H. (2007). Entity ranking on graphs: studies on expert finding. *Technical Report TR-CTIT*, 7-81.

Rossum, V., Guido., and Drake, F. (1995). Python tutorial. *Centrum voor Wiskunde en Informatica Amsterdam*, The Netherlands.

Zhu, X., Lyu, C., & Ji, D. (2020). Keyphrase Generation With CopyNet and Semantic Web. *IEEE Access*, *8*, 44202-44210. doi: 10.1109/access.2020.2977508

Sievert, C., & Shirley, C. (2014). LDAvis: A method for visualizing and interpreting topics. *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces, 63-70*.

Sinha, A., Cazabet, R., Vaudaine., R. (2018). Systematic Biases in Link Prediction: comparing heuristic and graph embedding based methods. *Complex networks 2018 - The 7th International Conference on Complex Networks and Their Applications*.

*Sun, Y., Barber, R., Gupta, M., Aggarwal, C. & Han, J. (2011). Co-author Relationship Prediction in Heterogeneous Bibliographic Networks. In: *International Conference on Advances in Social Networks Analysis and Mining. IEEE Computer Society*, 121-128.

**Sun, Y., Han, J., Yu, P., & Wu, T. (2011). PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. In: *Proceedings on Very Large Data Bases (VLDB).*

Shakibian, H., Charkari, N. and Jalili, S. (2016). A multilayered approach for link prediction in heterogeneous complex networks. *Journal of Computational Science*, 17, 73-82.

Shakibian, H., and Charkari, N. (2018). Statistical similarity measures for link prediction in heterogeneous complex networks. *Physica A*, 501, 248-263.

Tantardini, M., Leva, F., Tajoli, L. & Piccardi, C. (2019). Comparing methods for comparing networks. *Sci Rep*. 9,1, 17557.

Tight, M. (2014). Working in separate silos? What citation patterns reveal about higher education research internationally. *Higher education*, 68, 379-395.

Wang, Z., & Zhu.,J. (2014). Homophily versus preferential attachment: Evolutionary mechanisms of scientific collaboration networks. *International Journal of Modern Physics*, 25, 5.

Xu, J., Shen, S., Li, D., & Fu, Y. (2018). A Network-embedding Based Method for Author Disambiguation. *Proceedings of the 27th ACM International Conference of Information and Knowledge Management*. Torino: CIKM, 1735-1738.

Yuxiao, D., Nitesh, C., & Ananthram, S. (2017). Metapath2vec: Scalable Representation Learning for Heterogeneous Networks. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD),* 135-144. doi: 10.1145/3097983.3098036

Zhou, H., Sun, J., Zhao, Z., Yang, Y., Xie, A., & Chiclana, F. (2019). Attention-Based Deep Learning Model for Predicting Collaborations Between Different Research Affiliations. *IEEE Access*. 7, 118068-118076.

8.0 Appendix 1: Additional Discussion and Recommendations

8.1 Recommendations

- ***Extraction***: To improve on extraction performance other bibliographic data sources could be incorporated into the extraction pipeline, such as *Google Scholar*. A key issue we found with our extraction method was the length of time required to extract researcher profiles for each separate bibliographic pipeline – primarily, Scopus and ORCID – potentially the application of parallel computing in the cloud and further optimisation of extraction pipelines could be leveraged in addition to using commercial rather than public access APIs, in obtaining faster publication profile retrieval and improved efficiency. **See post-script for further discussion of Scopus's publication coverage in our results.*
- ***Disambiguation and Researcher Consolidation***: As Xu et al's (2018) method is highly adaptable, further network layers could be added during the disambiguation process, with some examples being citation relations, publication year and geography. Furthermore, hyperparameter tuning of the disambiguation algorithm to find the parameters that best optimize classification performance in terms of F1-score, could reduce the occurrence of false positive publications amongst researchers, as opposed to our current reliance on default parameter values used by Xu et al (2018). Furthermore, utilising DOI identifiers in research organisation databases for autonomous grid-searching of the best performing parameters would allow for this process to be incorporated into the overall tool pipeline.
- ***Research Topic Modelling***: Because our method utilises a fixed suboptimal corpus of cancer-related publications sourced outside of our toolkit, in order to more comprehensively capture researchers breadth of expertise amongst publications and, critically, to have the capacity to firstly provide a pipeline for obtaining a training corpus, it would be necessary to incorporate a more dynamic means of extracting training set publications for language modelling over any type of research organisation. A proposed method would be to utilise researchers publication keywords extracted in stage one 'test' publications to then iteratively extract **N** additional publications for each extracted researcher publication using the respective bibliographic repository, thereby acting as a latent representation around primary researchers original extracted papers, with N being a proportionally larger number than primary researcher publications in order to capture breadth in topic diversity and satisfactory corpus size for effective training of the topic model. Furthermore, these modelled topics over publications could potentially be used during stage two, adding a topic layer to our disambiguation method to further fine-tune the disambiguation model (Xu et al, 2018). In other words, keywords assigned in bibliographic databases such as PubMed could be used to "bootstrap" the extraction model training process.
- ***Research Collaboration Recommendations:*** Our research recommendation and link prediction method returned excellent results in predicting randomly sampled positive edges from our static research collaboration network. However, to generalise these recommendations more practically to the research context, evaluation of research collaboration predictions should be undertaken on a dynamic graph, using publication year to allow for dataset splitting according to time stamps, thereby capturing and

evaluating publication trends and predictions over time and moving away from a predictive model influenced by inferred predictions from surrounding residual network structure surrounding sampled and removed test-case positive edges (Sinha, Cazabet & Vaudaine, 2018).

- ***Research Collaboration Network Visualisation:*** The inclusion of other bibliometric information in earlier stages of our tool, would allow for more in-depth network visualisations, providing further layers and interactions between layers at the level of geography, citation relationships and development over time of these research networks; potentially serving as a useful means of capturing and sharing the progression of research collaboration over departments and centres of a research organisation.
- ***Graphical User-Interface (GUI) leveraging Rshiny and Reticulate:*** It is required to uphaul the disambiguation algorithm in stage two to Python 3, allowing fusion of R and python pipelines using Reticulate into a fully functional and deployed, user-friendly, front-end to back-end Rshiny Application.
- ***Scalable deployment:*** Once a functional Rshiny application through Reticulate is established, a R-server cloud platform such as *Rshinyapps.io* could allow for delivery of this service online to research organisations as a developed, accessible and scalable research collaboration tool (Allare et al, 2017).
- ***A broader perspective:*** With the prior suggestion of delivering such a tool within an established Rshiny cloud server, research organisations could essentially 'map' their separate department and research centre research networks as described throughout this technical report, using our infrastructure online to establish a pipeline from remote servers to organisation user databases to automatically update their researcher databases, model researcher topic expertise between departments and organisation partners and further store these research collaboration networks and trained topic models within this application cloud server for incrementally improved recommendations, expertise modelling capacity and in-depth network visualisations, iteratively growing these representations within remote SQL and NoSQL databases siloed by organisations and with key functionality such as research recommendations and topic modelling iteratively batch trained and improved within this Rshinyapp server by each organisational user for sequentially improved performance among and between research ecosystems. Furthermore, as more research bodies of their organisation and organisation research partners were to use such services more levels of research collaboration and topic diversity could be captured, thereby allowing more accurately modelled research expertise in the organisational research community and establishing a strong foundation for capturing the research collaboration network of the organisation and partners. With separate bodies of these organisations 'mapped' in terms of established publication output patterns, collaborations and topic expertise, research recommendation could be significantly more personalised within the organisations and between partners, with network visualisations better positioned to capture the departmental and interorganisational relationships of the collaborative research environment and thereby effectively using these technologies to foster research collaboration both at the organisation and at the broader interorganisational research collaboration level.

8.2 Scopus Post-script

In our results and evaluation of stage one to three – *extraction*, *disambiguation*, and *research profile consolidation* – it was observed that the Scopus extraction only provided coverage for a single researcher, due to obscure technical issues with the code used to address the Scopus API. Following evaluation of these results this error has been fixed, with *Figure 28* and *Table 15* displaying the corrected extraction results for primary researchers had this Scopus pipeline been fully functional, evidently resulting in far wider coverage than our prior results suggested and amongst our four bibliographical pipelines revealing Scopus to have a significant proportion of publication coverage in terms of research publications extracted against other pipelines, particularly for those researchers (namely *Norris*) displaying poor extraction performance in our prior Output 1 results. It is thereby quite likely the research publication profile extraction of stage one to three would perform significantly better in terms of true positives and false negatives due to this significantly greater publication coverage. This suggests the need for revaluation of this functionality of our tool for the quantification of the improvement in performance gained from full functionality of Scopus during our extraction phase, and thereby allowing for determination of the effect on false positive and false negative extraction proportions and overall F1-score extraction performance. Unfortunately, time did not permit such a re-evaluation.

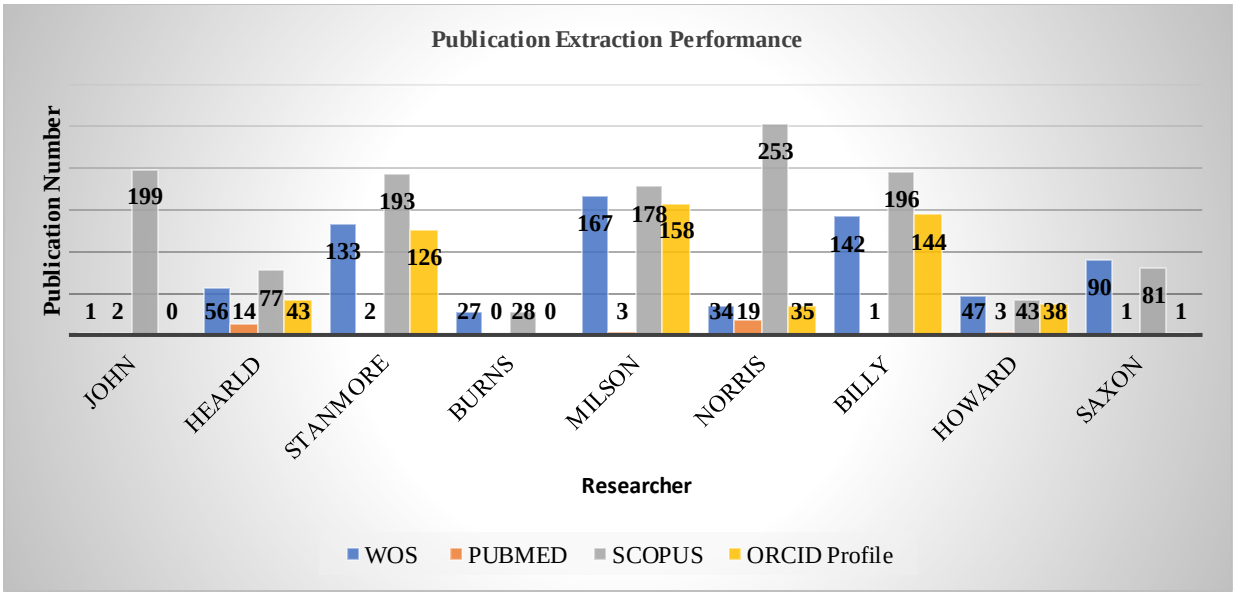

Figure 28 – Updated clustered histogram of researcher publication extraction performance

Researcher	WOS	PUBMED	SCOPUS	ORCID Profile
JOHN	1	2	199	0
HEARLD	56	14	77	43
STANMORE	133	2	193	126
BURNS	27	0	28	0
MILSON	167	3	178	158
NORRIS	34	19	253	35
BILLY	142	1	196	144
HOWARD	47	3	43	38
SAXON	90	1	81	1
Grand Total	**697**	**45**	**1248**	**545**

Table 15 – Updated table of researcher publication extraction performance

9.0 Appendix 2: Project Code

9.1 Researcher Profile Extraction

Stage one consists of four R scripts commencing the extraction of researchers through three initial pipelines (SCOPUS, PubMed & Web-of-Science) and finally, merging these datasets into a single ambiguous researcher publication dataset for stage two-three disambiguation & consolidation. This would involve a input section within the final Rshiny application for users to select their input file (Excel, Txt etc) containing their researchers ids and organisation publication records to then commence extraction. Furthermore, two adjustments in future would be to have a fourth pipeline for ORCID extraction directly at this stage (and thus removing the assumption that all ORCID publication are correct – as they are currently added post-disambiguation) and furthermore, the addition of the researcher training publication keyword search algorithm to compliment extraction of researcher publications (test) with a greater proportion and thus time for extraction of keywords on researchers publications to create a training corpus for the stage four topic model. The Rshiny application would provide a means of giving rough estimates for extraction times, however, likely it would be slow. To upscale these records would be loaded into an SQL database. While during stage five research network construction they would later be uploaded into a Neo4j database. Additionally, to allow for Reticulate to combine R and Python script functionality, this disambiguation algorithm (currently in Python 2 and using Networkx 1) would require uphauling to Networkx version2 and Python 2 allowing for a single Python Virtual Environment to be used in the application).

Script 1.1 Extract Scopus Researcher Data using R Programming Language and RStudio

---

```
#Importing Libraries:
library(easyPubMed)
library(dplyr)
library(ggplot2)
library(skimr)
library(rorcid)
library(dplyr)
library(igraph)
library(visNetwork)
library(stringr)
library(rorcid)
library(rscopus)
library(wosr)

# Get session ID using IP-Address (Note: Wifi-IP needs to be at an institution account who has access to Web of Science)
sid <- auth(NULL, NULL)

#Creating Extraction Summary:
Extraction_summary_df <- data.frame(Name=NA,
SCOPUS_Total=NA,SCOPUS_ID_Successful=NA,ORCID_ID_Successful=NA, OVERALL_SCOPUS_STATUS=NA)

#Moving to correct directory:
dir <- "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\University_Input_datasets"
setwd(dir)
```

```
#Loading and cleaning UNSW Researchers Dataset from provided Excel file:
UTS_researchers <- read.csv("TEST_10_SAMPLE_data_RESEARCHERS.csv",
                            header = TRUE,
                            quote="\"",
                            stringsAsFactors= TRUE,
                            strip.white = TRUE)
UTS_researchers$name_clean <- sapply(1:nrow(UTS_researchers),function(y){

  lname <- str_to_upper(str_trim(UTS_researchers$SURNAME[y]))
  lname_clean <- str_split(lname, " |-|'")
  lname_clean <- paste(lname_clean[[1]], sep="",collapse="")
  fname <-str_sub(str_to_upper(str_trim(UTS_researchers$FIRST_NAME[y])),1,1)
  return(paste(lname_clean,fname,sep=",",collapse=","))
})


#Loading in UNSW Cohort Publication Validation Dataset from provided Excel File:
UTS_PUBLICATIONS <- read.csv("TEST_10_SAMPLE_data_PUBS_MASTER.csv",
                             header = TRUE,
                             quote="\"",
                             stringsAsFactors= TRUE,
                             strip.white = TRUE)

#cleaning for publications validation:
for (row in 1:nrow(UTS_PUBLICATIONS))

{
  if (UTS_PUBLICATIONS$RE_flag[row] != "no"){
    re_key = UTS_PUBLICATIONS$RE_flag[row]
    test <- str_trim(str_replace(as.character(UTS_PUBLICATIONS$ALL_AUTHORS[row]), "-
|'|\\*",""))
    authors = str_extract_all(as.character(test), "\\w+, \\w+")
  }
  if (UTS_PUBLICATIONS$RE_flag[row] == "no"){
    test <- str_trim(str_replace(as.character(UTS_PUBLICATIONS$ALL_AUTHORS[row]), "-
|'|\\*",""))
    between_split_key = as.character(UTS_PUBLICATIONS$ALL_COAUTHORS_KEY[row])
    authors = str_split(as.character(test), between_split_key)
  }
  within_AUTH_key = UTS_PUBLICATIONS$within_AUTH_key[row]
  if (within_AUTH_key == ""){
    within_AUTH_key = " "
  }
  name_schema <- as.character(UTS_PUBLICATIONS$name_schema[row])
  list_index <- 0
  cleaned_author_list <- list()
  cleaned_author_list[1] = ""
  for (i in 1:length(authors[[1]])){
    name = str_split(authors[[1]][i], as.character(within_AUTH_key))
    index <- 0
```

```
    cleaned_name <- list()
    cleaned_name[[1]] = " "
    for (j in 1:length(name[[1]])){
      if (name[[1]][j] != ""){
        index <- index + 1
        cleaned_name[[1]][index] <- name[[1]][j]
      }
    }
    name <-  cleaned_name
    if (name_schema == "last,first"){
      lname <- name[[1]][1]
      fname <-  substr(name[[1]][length(name[[1]])],1,1)
      cleaned <- paste(str_to_upper(str_trim(lname)),str_to_upper(str_trim(fname)),
collapse=",", sep=",")
    }
    if (name_schema == "first,last")
      { lname <- name[[1]][length(name[[1]])]
        fname <- substr(name[[1]][1],1,1)
        cleaned <- paste(str_to_upper(str_trim(lname)),str_to_upper(str_trim(fname)),
collapse=",", sep=",")
    }
    list_index <- list_index + 1
    cleaned_author_list[[1]][list_index] = cleaned
    }
  cleaned_author_list <- paste(unlist(cleaned_author_list), collapse="; ")
UTS_PUBLICATIONS$cleaned_author_list[row] <- cleaned_author_list
}


#Subsetting for Researchers at UTS Cohort who have a Scopus ID (inclusive of duplicates):
UNSW_SCOPUS_SUBSET <- UTS_researchers[((UTS_researchers$SCOPUS.ID !=
'')&(!is.na(UTS_researchers$SCOPUS.ID)))|((UTS_researchers$ORCID.ID !=
'')&(!is.na(UTS_researchers$ORCID.ID))),]
UNSW_SCOPUS_SUBSET$SCOPUS.ID <- as.character(UNSW_SCOPUS_SUBSET$SCOPUS.ID)
print(paste("Initial amount of UTS researchers in validation dataset
is",length(UTS_researchers$SCOPUS.ID),
            "With only", length(UNSW_SCOPUS_SUBSET$SCOPUS.ID), "of these researchers with
Scopus IDs OR a supplementary ORCID ID"))

#This will be our vector of scopus ID researchers for data extraction:
SCOPUS_vector <- as.character(UTS_researchers$SCOPUS.ID)
NAME_vector <- UTS_researchers$SURNAME
ORCID_vector <- as.character(UTS_researchers$ORCID.ID)

#Having accessed Scopus website and requested API authorisation key, which we have saved
and stored implicitly we will first test to ensure this key is present prior to commencing
extraction:

key = get_api_key()
if (have_api_key())
{print("Have API Key")} else {"Failed: No API Key detected"}
```

```
#Getting paper dois for author:
get_doi <- function(x){
  #  This pulls the DOIs out of the ORCiD record:
  list.x <- x$df$`prism:doi`
  #  Catching objects with NULL DOI information:
  do.call(rbind.data.frame,lapply(list.x, function(x){

    if(is.na(x)){data.frame(value=NA)}
    else
    {data.frame(value = x)}}
  ))
}
#Function to pull affiliations for authors in get_papers:
getaffil <- function(y, x){
  auth_affil_ids <-
as.array(x$full_data$author[x$full_data$author$entry_number==y,]$`afid.$`)
  if (length(auth_affil_ids) != 0){

    for (i in 1:length(auth_affil_ids)){
      if (is.na(auth_affil_ids[i]) | auth_affil_ids[i] == ''| auth_affil_ids[i] == ' ')
{auth_affil_ids[i] <- NA}
      else{
        check <- x$full_data$affiliation[x$full_data$affiliation$afid == auth_affil_ids[i]
& x$full_data$affiliation$entry_number == y,  "affilname"]
      if (length(check) > 0){
        if (!check == ''& !check == ' ' &!is.na(check)) {auth_affil_ids[i]  <- check}}
      else {auth_affil_ids[i] <- NA}
      }}
    return(auth_affil_ids)
  }
  else {return(NA)}}


#Function to pull countries for authors in get_papers: NOTE: Countries not yet used in
official pipeline; see Appendix 1: Additional Discussion for updates to our model using
countries in disambiguation, recommendations and vizualisations.
getcountry <- function(y, x){
  auth_affil_ids <-
as.array(x$full_data$author[x$full_data$author$entry_number==y,]$`afid.$`)
  if (length(auth_affil_ids) != 0){

    for (i in 1:length(auth_affil_ids)){
      if (is.na(auth_affil_ids[i]) | auth_affil_ids[i] == ''|auth_affil_ids[i] == ' ')
{auth_affil_ids[i] <- NA}
      else{check <- x$full_data$affiliation[x$full_data$affiliation$afid ==
auth_affil_ids[i] & x$full_data$affiliation$entry_number == y,  "affiliation-country"]
      if (length(check) > 0){
        if (!check == ''& !check == ' ' &!is.na(check)) {auth_affil_ids[i]  <- check}}
      else {auth_affil_ids[i] <- NA}
      }}
```

```
    return(auth_affil_ids) }
  else {return(NA)}}

getabstract <- function(y, paper.doi, papers){
  abstract = abstract_retrieval(paper.doi[[y]][[1]], identifier = "doi")
  if (length(abstract$content) != 0) {
  if (length(abstract$content$`abstracts-retrieval-response`$item$bibrecord$head$abstracts)
!= 0)
  {return(abstract$content$`abstracts-retrieval-response`$item$bibrecord$head$abstracts)}
  else {abstract2 <- abstract_retrieval(papers[y,][4], identifier = "scopus_id")
  if (length(abstract2$content$`abstracts-retrieval-
response`$item$bibrecord$head$abstracts)!= 0){
    return(abstract2$content$`abstracts-retrieval-response`$item$bibrecord$head$abstracts)}
  else if ((length(abstract2$content$`abstracts-retrieval-
response`$item$bibrecord$head$abstracts)== 0))
  {return(NA)}}}
  else {return(NA)}
  }

#Getting pubmed ids for later PubMed data extraction for author based on papers
get_pubid <- function(x){
  #  This pulls the DOIs out of the ORCiD record:
  list.x <- x$df$`pubmed-id`
  if (is.null(list.x)){list.x <-  rep(NA, length(x$df$`dc:title`))}
  #  Catching NULL objects with missing DOI information:
  do.call(rbind.data.frame,lapply(list.x, function(x){
    if(is.na(x)){
      data.frame(value=NA)}
    else
    {data.frame(value = x)}}
  ))}

#Getting scopus identifier ids for documents in case DOI for papers are missing thus
preventing co-authors and papers being missed.
get_dc_scop_id <- function(x){
  #  This pulls the DOIs out of the ORCiD record:
  list.x <- x$df$`dc:identifier`
  if (is.null(list.x)){list.x <-  rep(NA, length(x$df$`dc:title`))}
  #  Catching objects with NULL DOI information:
  do.call(rbind.data.frame,lapply(list.x, function(x){
    if(is.na(x)){
      data.frame(value=NA)}
    else
    {data.frame(value = x)}}
  ))}

#Getting an author's papers and all co-authors names and papers:

#Extracting coverdate column as publication date (*Note: Not used in current model, however
in moving to dynamic graph for recommendations would allow for predictions and
visualisations overtime.
```

```
#The main function for SCOPUS researcher publication profile extraction:
get_papers <- function(x){
  papers <- data.frame(title = x$df$`dc:title`,
                       doi   = get_doi(x),
                       pubmedid = get_pubid(x),
                       scop_dcID = get_dc_scop_id(x))
  paper.doi <- lapply(1:nrow(papers), function(y){
    if(length(papers!=0)){
      if(!is.na(papers[y,2]))
      {return(as.character(check_dois(papers[y,2][1])$good))}
      else if (is.na(papers[y,2]) & !is.na(papers[y,4]))
      {return(list(as.character(papers[y,4])))}
            else
      {return(NA)}}})
  your.papers <- lapply(1:length(paper.doi), function(y){
    if(length(paper.doi[[y]]) == 0 | is.na(paper.doi[[y]])){
      data.frame(doi=NA, title=NA,abstract =NA,Venue=NA,scopus=NA, firstname=NA,surname=NA
,initials=NA, pubmedid = NA,affil=NA, country = NA, year = NA)
    } else {
                data.frame(doi = paper.doi[[y]][[1]],
                 title = as.character(x$df[x$df$entry_number == y, c("dc:title")]),
                 abstract = getabstract(y, paper.doi, papers),
                 Venue = as.character(x$df[x$df$entry_number == y,
c("prism:publicationName")]),
                 scopus = as.array(x$full_data$author[x$full_data$author$entry_number == y,
c("authid")]),
                 firstname =
as.array(paste(x$full_data$author[x$full_data$author$entry_number == y, c("given-name")])),
                 surname = as.array(x$full_data$author[x$full_data$author$entry_number ==
y, c("surname")]),
                 initials = as.array(x$full_data$author[x$full_data$author$entry_number ==
y, c("initials")]),
                 pubmedid = as.character(papers[y,3]),
                 affil =  getaffil(y, x),
                 country = getcountry(y, x),
                 year =  as.character(x$df[x$df$entry_number == y, c("prism:coverDate")]),
                 stringsAsFactors = FALSE)
    }})
      do.call(rbind.data.frame, your.papers)
}

#Firstly, we will get the papers (get_papers) for all researchers that have UNIQUE scopus
IDs from our scopus validation dataset vector.
print(paste("We have", length(SCOPUS_vector), "unique researchers with scopus IDs -
commencing data extraction"))

#Extracting co-authors of each primary researcher and concatenating into a list of co-
authors for each primary author, DOIs, countries, publication dates, titles, abstracts and
scopus_id - this will thereby get all first-order networks of each researcher.
all.coauthors_first_order <- list()
```

```
for (i in 1:length(UTS_researchers$SURNAME))
  { cat(i, "of", length(SCOPUS_vector), "scopus=", SCOPUS_vector[i], "\n")
  if (!is.na(SCOPUS_vector[i]) & !SCOPUS_vector[i]=='' & !SCOPUS_vector[i]=="\"\"" )
  {
    print("Scopus i not NA, beginning scopus ID extraction")
    test <- author_data(au_id=SCOPUS_vector[i])
    if (length(test$entries) > 1)
    {print(paste("NON-NA SCOPUS ID - Entered:length(test$entries > 1)",
length(test$entries)))
      all.coauthors_first_order[i] <- list(get_papers(test))
      print("SCOPID Extraction was successful.")
      Extraction_summary_df[i, ] <-
c(UTS_researchers$name_clean[i],length(unique(all.coauthors_first_order[i][[1]]$doi)),"SUCC
ESSFUL","NOT-ATTEMPTED","SUCCESSFUL")
      }
    else if (length(test$entries) == 1)
    {print(paste("NON-NA SCOPUS ID - Entered:length(test$entries == 1)",
length(test$entries)))
      print(paste("SCOPID Extraction was NOT successful, attempting ORCID ID input into
SCOPUS API as alternative, using", ORCID_vector[i]))
      test <- author_data(au_id=ORCID_vector[i], searcher='ORCID')
      if (length(test$entries) > 1)
      {all.coauthors_first_order[i] <- list(get_papers(test))
      print("Successfully extracted using ORCID ID")
      Extraction_summary_df[i, ] <-
c(UTS_researchers$name_clean[i],length(unique(all.coauthors_first_order[i][[1]]$doi)),"UNSU
CCESSFUL","SUCCESSFUL","SUCCESSFUL")
      }
      else if (length(test$entries)==1)
      {print(paste("ORCID extraction unsuccessful making", i, "NA dataframe"))
        all.coauthors_first_order[i] <- list(data.frame(doi=NA, title=NA,abstract
=NA,Venue=NA,scopus=NA, firstname=NA,surname=NA ,initials=NA, pubmedid = NA,affil=NA,
country = NA,year=NA))
        Extraction_summary_df[i, ] <-
c(UTS_researchers$name_clean[i],0,"UNSUCCESSFUL","UNSUCCESSFUL","UNSUCCESSFUL")
}
      }}

  else if (is.na(SCOPUS_vector[i]) |SCOPUS_vector[i]=='')
  {print(paste(i, "SCOPUS", SCOPUS_vector[i], "is NA will attempt using ORCID ID in SCOPUS
API instead:", ORCID_vector[i]))
    if (!is.na(ORCID_vector[i]) & nchar(ORCID_vector[i])==19)
    {
      test <- author_data(au_id=ORCID_vector[i], searcher='ORCID')
      if (length(test$entries) > 1) {all.coauthors_first_order[i] <- list(get_papers(test))
      print("Successfully extracted using ORCID ID in SCOPUS")
      Extraction_summary_df[i, ] <-
c(UTS_researchers$name_clean[i],length(unique(all.coauthors_first_order[i][[1]]$doi)),"NOT-
PROVIDED","SUCCESSFUL","SUCCESSFUL")}
      else if (length(test$entries)==1)
```

```
      { #print(paste("ORCID extraction unsuccessful making attempting using ORCID ID in
Web-Of-Science API"))
        print(paste("ORCID extraction unsuccessful making", i, "NA dataframe"))
        all.coauthors_first_order[i] <- list(data.frame(doi=NA, title=NA,abstract
=NA,Venue=NA,scopus=NA, firstname=NA,surname=NA ,initials=NA, pubmedid = NA,affil=NA,
country = NA,year=NA))
        Extraction_summary_df[i, ] <- c(UTS_researchers$name_clean[i],0,"NOT-
PROVIDED","UNSUCCESSFUL","UNSUCCESSFUL")
      }   }
    else {print(paste("WARNING: NONE CAPTURED BY SCOPUS EXTRACT OR SCOPUS USING ORCID ID OR
WOS/PUBMED EXTRACTS DEFAULTING TO NA DATAFRAME"))
      all.coauthors_first_order[i] <- list(data.frame(doi=NA, title=NA,abstract
=NA,Venue=NA,scopus=NA, firstname=NA,surname=NA ,initials=NA, pubmedid = NA,affil=NA,
country = NA,year=NA))
      Extraction_summary_df[i, ] <- c(UTS_researchers$name_clean[i],0,"NOT-PROVIDED","NOT-
PROVIDED","UNSUCCESSFUL")}
  }
}
#Moving to correct directory to save Extraction Summary for SCOPUS:
dir <- "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data"
setwd(dir)
Extraction_summary_df_SCOPUS <- Extraction_summary_df
save(Extraction_summary_df_SCOPUS, file = "Extraction_summary_df_SCOPUS_v1.Rdata")
all.coauthors_first_order_SCOPUS_EVALv1 <- all.coauthors_first_order
#Saving extracted SCOPUS publication dataset for all primary researchers:
save(all.coauthors_first_order_SCOPUS_EVALv1, file =
"all.coauthors_first_order_SCOPUS_EVALv1.Rdata")
```

Script 1.2 Extract PubMed Researcher Data using R Programming Language and RStudio

```
#Import required libraries:
library(easyPubMed)
library(dplyr)
library(ggplot2)
library(skimr)
library(rorcid)
library(dplyr)
library(igraph)
library(visNetwork)
library(stringr)

#Importing UNSW Based Dataset from Excel and creating extraction summary:
Extraction_summary_df <- data.frame(Name=NA, PUBMED_Total=NA,PUBMED_ORCID_Successful=NA,
OVERALL_PUBMED_STATUS=NA)
#Moving to correct directory:
dir <- "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\University_Input_datasets"
setwd(dir)
#Loading in the UNSW Researchers Dataset:
UTS_researchers <- read.csv("TEST_10_SAMPLE_data_RESEARCHERS.csv",
                            header = TRUE,
                            quote="\"",
                            stringsAsFactors= TRUE,
                            strip.white = TRUE)
UTS_researchers$name_clean <- sapply(1:nrow(UTS_researchers),function(y){
  lname <- str_to_upper(str_trim(UTS_researchers$SURNAME[y]))
  lname_clean <- str_split(lname, " |-|'")
  lname_clean <- paste(lname_clean[[1]], sep="",collapse="")
  fname <-str_sub(str_to_upper(str_trim(UTS_researchers$FIRST_NAME[y])),1,1)
  return(paste(lname_clean,fname,sep=",",collapse=","))
})

#Loading in Validation Dataset:
UTS_PUBLICATIONS <- read.csv("TEST_10_SAMPLE_data_PUBS_MASTER.csv",
                             header = TRUE,
                             quote="\"",
                             stringsAsFactors= TRUE,
                             strip.white = TRUE)
#Cleaning for publications valdiation:
for (row in 1:nrow(UTS_PUBLICATIONS))
{
  if (UTS_PUBLICATIONS$RE_flag[row] != "no"){
    re_key = UTS_PUBLICATIONS$RE_flag[row]
    #UTS_PUBLICATIONS$ALL_AUTHORS[row] <-
str_replace(as.character(UTS_PUBLICATIONS$ALL_AUTHORS[row]), "-|'","")
    test <- str_trim(str_replace(as.character(UTS_PUBLICATIONS$ALL_AUTHORS[row]), "-
|'|\\*",""))
    authors = str_extract_all(as.character(test), "\\w+, \\w+")
```

```
  }
  if (UTS_PUBLICATIONS$RE_flag[row] == "no"){
    test <- str_trim(str_replace(as.character(UTS_PUBLICATIONS$ALL_AUTHORS[row]), "-
|'|\\*",""))
    between_split_key = as.character(UTS_PUBLICATIONS$ALL_COAUTHORS_KEY[row])
    authors = str_split(as.character(test), between_split_key)
  }
  within_AUTH_key = UTS_PUBLICATIONS$within_AUTH_key[row]
  if (within_AUTH_key == ""){
    within_AUTH_key = " "
  }
  name_schema <- as.character(UTS_PUBLICATIONS$name_schema[row])
  list_index <- 0
  cleaned_author_list <- list()
  cleaned_author_list[1] = ""
  for (i in 1:length(authors[[1]])){
    name = str_split(authors[[1]][i], as.character(within_AUTH_key))
    index <- 0
    cleaned_name <- list()
    cleaned_name[[1]] = " "
    for (j in 1:length(name[[1]])){
      if (name[[1]][j] != ""){
        index <- index + 1
        cleaned_name[[1]][index] <- name[[1]][j]
      } }
    name <-  cleaned_name
    if (name_schema == "last,first"){
      lname <- name[[1]][1]
      fname <-  substr(name[[1]][length(name[[1]])],1,1)
      cleaned <- paste(str_to_upper(str_trim(lname)),str_to_upper(str_trim(fname)),
collapse=",", sep=",")  }
    if (name_schema == "first,last")
    { lname <- name[[1]][length(name[[1]])]
    fname <- substr(name[[1]][1],1,1)
    cleaned <- paste(str_to_upper(str_trim(lname)),str_to_upper(str_trim(fname)),
collapse=",", sep=",")
    }
    list_index <- list_index + 1
    cleaned_author_list[[1]][list_index] = cleaned
  }
  cleaned_author_list <- paste(unlist(cleaned_author_list), collapse="; ")
  UTS_PUBLICATIONS$ALL_AUTHORS_CLEANED[row] <- cleaned_author_list
}
#Creating vector of all researchers ORCID IDs:
UNSW_ORCID_SUBSET <- UTS_researchers
UNSW_ORCID_SUBSET$ORCID.ID <- as.character(UNSW_ORCID_SUBSET$ORCID.ID)
print(paste("Initial amount of UTS researchers in validation dataset
is",length(UTS_researchers$ORCID.ID)))
#41 out of 46 researchers with ORCID ids

#Here is a character vector of all researcher of the UTS cohort with ORCID IDS
```

```
ORCID_COLUMN <- UNSW_ORCID_SUBSET$ORCID.ID
NAME_vector <-  UNSW_ORCID_SUBSET$SURNAME

Beginning main functions for PubMed Researcher Profile Extraction:
#Getting paper dois for author:
get_doi <- function(x){
  #  This pulls the DOIs out of the ORCiD record:
  list.x <- x$doi
  do.call(rbind.data.frame,lapply(list.x, function(x){
     if(is.na(x)){data.frame(value=NA)}
    else
    {data.frame(value = x)}}
  ))
}
#Getting pubmed ids for later PubMed data extraction for author based on papers
get_pubid <- function(x){
  #  This pulls the DOIs out of the ORCiD record:
  list.x <- x$pmid
  if (is.null(list.x)){list.x <-  rep(NA, length(x$title))}
  do.call(rbind.data.frame,lapply(list.x, function(x){
    if(is.na(x)){
      data.frame(value=NA)}
        else
          {data.frame(value = x)}}
  ))}
#The main extraction function for PubMed:
get_papers <- function(x){

  papers <- data.frame(title = x$title,
                       doi   = get_doi(x),
                       pubmedid = get_pubid(x))

  paper.doi <- lapply(1:nrow(papers), function(y){
    if(length(papers!=0)){
      if(!is.na(papers[y,2]) & !(papers[y,2] == ''))
      {return(as.character(check_dois(papers[y,2][1])$good))}
         else if ((is.na(papers[y,2])|(papers[y,2] == '')) & (!is.na(papers[y,3]) &
!(papers[y,3] == '') ))
      {return(list(as.character(papers[y,3])))}

      else
      {return(NA)}}})
    your.papers <- lapply(1:length(paper.doi), function(y){
    if(length(paper.doi[[y]]) == 0 | is.na(paper.doi[[y]])){
      data.frame(doi=NA, surname=NA, firstname=NA, pubmedid = NA, venue=NA,
affil=NA,title=NA,abstract=NA,month=NA,year=NA)
    } else {
      data.frame(doi = paper.doi[[y]][[1]],
                 surname = x$lastname,
                 firstname = x$firstname,
                 pubmedid = as.character(papers[y,3]),
```

```
                  venue = x$journal, #Journal
                  affil = x$address, #affil
                  title = x$title, #title
                  abstract = x$abstract,
                  month = x$month,
                  year = x$year,
                  stringsAsFactors = FALSE)}})
  do.call(rbind.data.frame, your.papers)
}

#Extracting co-authors of each researcher and concatenating into a list of co-authors for
each primary author, DOI, titles, abstracts, country, publication date, venues,
organisations, Pubmed ids - this will thereby get all first-order networks of each
researcher.
all.coauthors_first_order_PUBMED_DISAMB_v1 <- list()
for (i in 1:length(ORCID_COLUMN))
{ cat(i, "of", length(ORCID_COLUMN), "ORCID=", ORCID_COLUMN[i], "\n")
  if (!is.na(ORCID_COLUMN[i]) & (nchar(ORCID_COLUMN[i]) == 19) & !ORCID_COLUMN[i] ==
"\"\""){
    myQuery <- str_c(unlist(str_split(ORCID_COLUMN[i], '-')), collapse='')
    myIdList <- get_pubmed_ids(myQuery)
    if (myIdList$Count > 0)
    {
      t <- fetch_pubmed_data(pubmed_id_list = myIdList, retmax =myIdList$RetMax, retstart =
myIdList$RetStart)
      t2 <- table_articles_byAuth(t, included_authors = "all", getKeywords = TRUE)
      all.coauthors_first_order_PUBMED_DISAMB_v1[i] <- list(get_papers(t2))
      Extraction_summary_df[i, ] <-
c(UTS_researchers$name_clean[i],length(unique(all.coauthors_first_order_PUBMED_DISAMB_v1[i]
[[1]]$doi)),"SUCCESSFUL","SUCCESSFUL")
    }
    else {
          print(paste(i, "- Empty myIdList$IdList for", ORCID_COLUMN[i], "with Idlist value
of", myIdList$IdList, "& query string of:", myQuery))
        print("ORCID Unsuccessful, returning empty list for researcher")
        all.coauthors_first_order_PUBMED_DISAMB_v1[i] <- list(data.frame(doi=NA,
surname=NA, firstname=NA,  pubmedid = NA, venue=NA,
affil=NA,title=NA,abstract=NA,month=NA,year=NA))
        y <- y + 1
        Extraction_summary_df[i, ] <-
c(UTS_researchers$name_clean[i],0,"UNSUCCESSFUL","UNSUCCESSFUL")
       }
}
  }
  else
  {print("ORCID ID Unsuccessful, returning empty list for researcher")
    all.coauthors_first_order_PUBMED_DISAMB_v1[i] <- list(data.frame(doi=NA, surname=NA,
firstname=NA,  pubmedid = NA, venue=NA, affil=NA,title=NA,abstract=NA,month=NA,year=NA))
    Extraction_summary_df[i, ] <- c(UTS_researchers$name_clean[i],0,"NOT-
PROVIDED","UNSUCCESSFUL")
    y <- y + 1}}
```

```
#print(paste("Overall, there were", y, "ORCID queries that were rejected or did not return
matching PUBMED IDs AND alternatively could not use Web-of-Science/ORCID query"))
print(paste("Overall, there were", y, "ORCID queries that were rejected"))
all.coauthors_first_order_EVAL_v1 <- all.coauthors_first_order_PUBMED_DISAMB_v1
#Moving to correct directory:
dir <- "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data"
setwd(dir)
#Saving PubMed Extraction Summary Table and extracted PubMed dataset for each Primary
researcher using PubMed Ids and Orcid Ids:
Extraction_summary_df_PUBMED <- Extraction_summary_df
save(Extraction_summary_df_PUBMED, file = "Extraction_summary_df_PUBMED_v1.Rdata")
save(all.coauthors_first_order_EVAL_v1, file =
"all.coauthors_first_order_PUBMED_EVAL_v1.Rdata")
```

Script 1.3 Extract Web-of-Science Researcher Data using R Programming Language and RStudio

```
#Import libraries:
library(dplyr)
library(ggplot2)
library(skimr)
library(wosr)
library(dplyr)
library(igraph)
library(visNetwork)
library(stringr)
library(rorcid)

# Get session ID using IP-Address (Note: Wifi-IP needs to be at any institution who has
access to Web of Science account- university)
sid <- auth(NULL, NULL)
#Moving to correct directory:
dir <- "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\University_Input_datasets"
setwd(dir)
Extraction_summary_df <- data.frame(Name=NA,
WOS_Total=NA,WOS_ID_Successful=NA,WOS_ORCID_Successful=NA, OVERALL_WOS_STATUS=NA)


#Loading in the UNSW Researchers Dataset from provided Excel:
UTS_researchers <- read.csv("TEST_10_SAMPLE_data_RESEARCHERS.csv",
                            header = TRUE,
                            quote="\"",
                            stringsAsFactors= TRUE,
                            strip.white = TRUE)
UTS_researchers$name_clean <- sapply(1:nrow(UTS_researchers),function(y){
  lname <- str_to_upper(str_trim(UTS_researchers$SURNAME[y]))
  lname_clean <- str_split(lname, " |-|'")
  lname_clean <- paste(lname_clean[[1]], sep="",collapse="")
  print(lname_clean)
  fname <-str_sub(str_to_upper(str_trim(UTS_researchers$FIRST_NAME[y])),1,1)
  print(fname)
  return(paste(lname_clean,fname,sep=",",collapse=","))
})
#Loading in Researcher Publication Validation Dataset from Excel:
UTS_PUBLICATIONS <- read.csv("TEST_10_SAMPLE_data_PUBS_MASTER.csv",
                             header = TRUE,
                             quote="\"",
                             stringsAsFactors= TRUE,
                             strip.white = TRUE)
#Cleaning for publications:
for (row in 1:nrow(UTS_PUBLICATIONS))
{
  if (UTS_PUBLICATIONS$RE_flag[row] != "no"){
    re_key = UTS_PUBLICATIONS$RE_flag[row]
```

```
    #UTS_PUBLICATIONS$ALL_AUTHORS[row] <-
str_replace(as.character(UTS_PUBLICATIONS$ALL_AUTHORS[row]), "-|'","")
    test <- str_trim(str_replace(as.character(UTS_PUBLICATIONS$ALL_AUTHORS[row]), "-
|'|\\*",""))
        authors = str_extract_all(as.character(test), "\\w+, \\w+")
  }
  if (UTS_PUBLICATIONS$RE_flag[row] == "no"){
    test <- str_trim(str_replace(as.character(UTS_PUBLICATIONS$ALL_AUTHORS[row]), "-
|'|\\*",""))
    between_split_key = as.character(UTS_PUBLICATIONS$ALL_COAUTHORS_KEY[row])
    authors = str_split(as.character(test), between_split_key)
  }
  within_AUTH_key = UTS_PUBLICATIONS$within_AUTH_key[row]
  if (within_AUTH_key == ""){
    within_AUTH_key = " "
  }
  name_schema <- as.character(UTS_PUBLICATIONS$name_schema[row])
  list_index <- 0
  cleaned_author_list <- list()
  cleaned_author_list[1] = ""
  for (i in 1:length(authors[[1]])){
    name = str_split(authors[[1]][i], as.character(within_AUTH_key))
    index <- 0
    cleaned_name <- list()
    cleaned_name[[1]] = " "
    for (j in 1:length(name[[1]])){
      if (name[[1]][j] != ""){
        index <- index + 1
        cleaned_name[[1]][index] <- name[[1]][j]
      }}
    name <-  cleaned_name
    if (name_schema == "last,first"){
      lname <- name[[1]][1]
      fname <-  substr(name[[1]][length(name[[1]])],1,1)
      cleaned <- paste(str_to_upper(str_trim(lname)),str_to_upper(str_trim(fname)),
collapse=",", sep=",")
    }
    if (name_schema == "first,last")
    { lname <- name[[1]][length(name[[1]])]
    fname <- substr(name[[1]][1],1,1)
    cleaned <- paste(str_to_upper(str_trim(lname)),str_to_upper(str_trim(fname)),
collapse=",", sep=",")
    }
    list_index <- list_index + 1
    cleaned_author_list[[1]][list_index] = cleaned
  }
  cleaned_author_list <- paste(unlist(cleaned_author_list), collapse="; ")
  UTS_PUBLICATIONS$cleaned_author_list[row] <- cleaned_author_list
}
```

###Extracting Web of Science authors and papers from Web of science API using SPHERE

Dataset of Cohort UTS Researchers in Validation Dataset

```
#Based on the 'Gathering Bibliometric Information from the Scopus API using wosr' from the
Johns Hopkins Bloomberg School of Public Health authored by John Muschelli, 2018.

#Subsetting for Researchers at UTS Cohort who have a WOS ID (inclusive of duplicates):
UNSW_WOS_SUBSET <- UTS_researchers[((UTS_researchers$RESEARCHER.ID !=
'')&(!is.na(UTS_researchers$RESEARCHER.ID)))|
                                    ((UTS_researchers$ORCID.ID !=
'')&(!is.na(UTS_researchers$ORCID.ID))),]
UNSW_WOS_SUBSET$RESEARCHER.ID <- as.character(UNSW_WOS_SUBSET$RESEARCHER.ID)
print(paste("Initial amount of UTS researchers in validation dataset
is",length(UTS_researchers$RESEARCHER.ID),
            "With only", length(UNSW_WOS_SUBSET$RESEARCHER.ID), "of these researchers with
Researcher IDs OR ORCID IDS"))
#10 WOS researcher IDS or ORCiD IDs in total.

#This will be our vector of WOS ID researchers for data extraction and visualisation:
WOS_vector <- as.character(UTS_researchers$RESEARCHER.ID)
NAME_vector <- UTS_researchers$SURNAME
ORCID_vector <- as.character(UTS_researchers$ORCID.ID)

#Pull affiliation data for each author within get_papers function.
getaffil <- function(y, x, papers){
  if (!is.null(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no %in% x$author[x$author$ut == papers[y,2],2]),3]))
  {print('null')
    Addy_nums <- as.array(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no %in% x$author[x$author$ut == papers[y,2],2]),3])
    #Taking first affiliation if more than 1 affiliation for author (first by default) to
avoid error
    if (length(x$author[x$author$ut==papers[y,2],3]) != length(Addy_nums) &
length(Addy_nums)!=0 )
      #create a new list for addy_nums array:

    {
     #finding each author
      addy_nums2 <- list()
      for (i in 1:nrow(x$author[x$author$ut==papers[y,2],])) {
        row <- i #researcher
        author_no <-x$author[x$author$ut==papers[y,2],][row, 2]
        #Does each author have 1 or more addresses?
        if (nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),]) > 1)
          #Then researcher has more than 1 address - so we will take the first as default
        {
          test <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][1,2]
          test2 <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][2,2]
          if (length(x$address[x$address$ut == papers[y,2] & x$address$addr_no == test,
ADDRESS_COL] != 0))
          {
```

```
            addy_nums2[i] <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][1,2]}
          else if  (length(x$address[x$address$ut == papers[y,2] & x$address$addr_no ==
test2, ADDRESS_COL] != 0))
          { print(paste("test1 failed - trying to make addy_nums2[i] 4 test2:
",x$author_address[(x$author_address$ut == papers[y,2]) & (x$author_address$author_no ==
author_no),][2,2]))
            addy_nums2[i] <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][2,2]}
          else
          {
            print("failed test 2 - making NA for researcher with > 1 addresses")
            addy_nums2[i] <- NA}
        }
        #Otherwise, add in researchers only address to new address array:
        else if(nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),]) == 1) {
          print("Using resarchers only address, as nrows for author == 1")

          addy_nums2[i] <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][,2]}
        else if (nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),]) == 0 |
                 is.null(nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),])))
        {print("Making resarchers only address NA, as nrows for author not == 0 |
is.null(researchers address no.)")
          addy_nums2[i] <- NA }
      }
      #Replace address numbers array with updated address number array only taking first
address for each
      #author as default to correct unequal rows in get papers when extracting address
data.
      Addy_nums <- as.array(unlist(addy_nums2))
    }
    for (i in 1:length(Addy_nums))
    {
      if (length(Addy_nums) == 0 )
      {Addy_nums[i] <- NA}

      else {Addy <- x$address[x$address$ut == papers[y,2] & x$address$addr_no ==
Addy_nums[i], ADDRESS_COL]
      if (length(Addy) == 0)
      {Addy_nums[i] = NA}
      else
      {Addy_nums[i] = Addy}}

    }
    return(Addy_nums)}
  else { return(NA)}}
```

```
#Pull city data for each author within get_papers function.
getcity <- function(y, x, papers){
  if (!is.null(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no %in% x$author[x$author$ut == papers[y,2],2]),3]))
  {print('null')
    Addy_nums <- as.array(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no %in% x$author[x$author$ut == papers[y,2],2]),3])
    print("initial addy numbers are:")
    print(Addy_nums)

    #Taking first affiliation if more than 1 affiliation for author (first by default) to
avoid error
    if (length(x$author[x$author$ut==papers[y,2],3]) != length(Addy_nums) &
length(Addy_nums)!=0 )
      #create a new list for addy_nums array:

    {#find each author
      print("entered addy num 2")
      addy_nums2 <- list()
      for (i in 1:nrow(x$author[x$author$ut==papers[y,2],])) {
        row <- i #researcher

        author_no <-x$author[x$author$ut==papers[y,2],][row, 2]
        print("author_no is:")
        print(author_no)
        #Does each author have 1 or more addresses?
        if (nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),]) > 1)
          #than researcher has more then 1 address - so we will take the first as default
        {
          test <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][1,2]
          print(paste("test is:", test))

          test2 <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][2,2]
          print(paste("test2 is:", test2))
          if (length(x$address[x$address$ut == papers[y,2] & x$address$addr_no == test,
ADDRESS_COL] != 0))
          {
            print(paste("trying to make addy_nums2[i]:
",x$author_address[(x$author_address$ut == papers[y,2]) & (x$author_address$author_no ==
author_no),][1,2]))
            addy_nums2[i] <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][1,2]}
          else if  (length(x$address[x$address$ut == papers[y,2] & x$address$addr_no ==
test2, ADDRESS_COL] != 0))
          { print(paste("test1 failed - trying to make addy_nums2[i] 4 test2:
",x$author_address[(x$author_address$ut == papers[y,2]) & (x$author_address$author_no ==
author_no),][2,2]))
```

```
            addy_nums2[i] <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][2,2]}
          else
          {
            print("failed test 2 - making NA for researcher with > 1 addresses")
            addy_nums2[i] <- NA}
        }
        #Otherwise, add in researchers only address to new address array:
        else if(nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),]) == 1) {
          print("Using resarchers only address, as nrows for author == 1")

          addy_nums2[i] <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][,2]}
        else if (nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),]) == 0 |
                 is.null(nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),])))
        {print("Making resarchers only address NA, as nrows for author not == 0 |
is.null(researchers address no.)")
          addy_nums2[i] <- NA }
      }
      #Replace address numbers array with updated address number array only taking first
address for each
      #author as default to correct unequal rows in get papers when extracting address
data.
      Addy_nums <- as.array(unlist(addy_nums2))
      print("New addy numbers are")
      print(Addy_nums)
    }
    for (i in 1:length(Addy_nums))
    {
      print(Addy_nums[i])
      if (length(Addy_nums) == 0 )
      {Addy_nums[i] <- NA}

      else {Addy <- x$address[x$address$ut == papers[y,2] & x$address$addr_no ==
Addy_nums[i], ADDRESS_COL]
      print(paste("here",Addy))
      if (length(Addy) == 0)
      {Addy_nums[i] = NA}
      else
      {Addy_nums[i] = Addy}}

    }
    return(Addy_nums)}

  else {print("end")
    return(NA)}}

#Pull country data for each author within get_papers function.
```

```
getcountry <- function(y, x, papers){
  print("Beginning")
  ADDRESS_COL <- 7
  if (!is.null(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no %in% x$author[x$author$ut == papers[y,2],2]),3]))
  {print('null')
    Addy_nums <- as.array(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no %in% x$author[x$author$ut == papers[y,2],2]),3])
    print("initial addy numbers are:")
    print(Addy_nums)

    #Taking first affiliation if more than 1 affiliation for author (first by default) to
avoid error
    if (length(x$author[x$author$ut==papers[y,2],3]) != length(Addy_nums) &
length(Addy_nums)!=0 )
      #create a new list for addy_nums array:

    {#find each author
      print("entered addy num 2")
      addy_nums2 <- list()
      for (i in 1:nrow(x$author[x$author$ut==papers[y,2],])) {
        row <- i #researcher

        author_no <-x$author[x$author$ut==papers[y,2],][row, 2]
        #Does each author have 1 or more countries?
        if (nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),]) > 1)
          #than researcher has more than 1 country - so we will take the first as default
          #If so, take the first addresses only for this author
        {
          test <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][1,2]
          print(paste("test is:", test))

          test2 <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][2,2]
          print(paste("test2 is:", test2))
          if (length(x$address[x$address$ut == papers[y,2] & x$address$addr_no == test,
ADDRESS_COL] != 0))
          {
            print(paste("trying to make addy_nums2[i]:
",x$author_address[(x$author_address$ut == papers[y,2]) & (x$author_address$author_no ==
author_no),][1,2]))
            addy_nums2[i] <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][1,2]}
          else if  (length(x$address[x$address$ut == papers[y,2] & x$address$addr_no ==
test2, ADDRESS_COL] != 0))
          { print(paste("test1 failed - trying to make addy_nums2[i] 4 test2:
",x$author_address[(x$author_address$ut == papers[y,2]) & (x$author_address$author_no ==
author_no),][2,2]))
```

```
            addy_nums2[i] <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][2,2]}
          else
          {
            print("failed test 2 - making NA for researcher with > 1 addresses")
            addy_nums2[i] <- NA}
        }
        #Otherwise, add in researchers only country to new address array:
        else if(nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),]) == 1) {
          print("Using resarchers only address, as nrows for author == 1")

          addy_nums2[i] <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][,2]}
        else if (nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),]) == 0 |
                 is.null(nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),])))
        {print("Making resarchers only address NA, as nrows for author not == 0 |
is.null(researchers address no.)")
          addy_nums2[i] <- NA }
      }
      #Replace country numbers array with updated address number array only taking first
address for each
      #author as default to correct unequal rows in get papers when extracting address
data.
      Addy_nums <- as.array(unlist(addy_nums2))
      print("New addy numbers are")
      print(Addy_nums)
    }
    for (i in 1:length(Addy_nums))
    {
      print(Addy_nums[i])
      if (length(Addy_nums) == 0 )
      {Addy_nums[i] <- NA}

      else {Addy <- x$address[x$address$ut == papers[y,2] & x$address$addr_no ==
Addy_nums[i], ADDRESS_COL]
      print(paste("here",Addy))
      if (length(Addy) == 0)
      {Addy_nums[i] = NA}
      else
      {Addy_nums[i] = Addy}}

    }
    return(Addy_nums)}

  else {print("end")
    return(NA)}}

#Pull state data for each author within get_papers function.
```

```
getstate <- function(y, x, papers){
  print("Beginning")
  ADDRESS_COL <- 6
  if (!is.null(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no %in% x$author[x$author$ut == papers[y,2],2]),3]))
  {print('null')
    Addy_nums <- as.array(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no %in% x$author[x$author$ut == papers[y,2],2]),3])

  #Taking first state if more than 1 state for author (first by default) to avoid error
  if (length(x$author[x$author$ut==papers[y,2],3]) != length(Addy_nums) &
length(Addy_nums)!=0 )
    #create a new list for addy_nums array:

  {#find each author
    print("entered addy num 2")
    addy_nums2 <- list()
    for (i in 1:nrow(x$author[x$author$ut==papers[y,2],])) {
      row <- i #researcher
      author_no <-x$author[x$author$ut==papers[y,2],][row, 2]
      print("author_no is:")
      print(author_no)
      #Does each author have 1 or more states?
      if (nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),]) > 1)
        #than researcher has more then 1 states - so we will take the first as default
        #If so, take the first addresses only for this author
      {
        test <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][1,2]
        print(paste("test is:", test))

        test2 <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][2,2]
        print(paste("test2 is:", test2))
        if (length(x$address[x$address$ut == papers[y,2] & x$address$addr_no == test,
ADDRESS_COL] != 0))
        {
          print(paste("trying to make addy_nums2[i]:
",x$author_address[(x$author_address$ut == papers[y,2]) & (x$author_address$author_no ==
author_no),][1,2]))
          addy_nums2[i] <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][1,2]}
        else if  (length(x$address[x$address$ut == papers[y,2] & x$address$addr_no ==
test2, ADDRESS_COL] != 0))
        { print(paste("test1 failed - trying to make addy_nums2[i] 4 test2:
",x$author_address[(x$author_address$ut == papers[y,2]) & (x$author_address$author_no ==
author_no),][2,2]))
          addy_nums2[i] <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][2,2]}
        else
```

```
        {
          addy_nums2[i] <- NA}
      }
      #Otherwise, add in researchers only state to new state array:
      else if(nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),]) == 1) {
        print("Using resarchers only address, as nrows for author == 1")

        addy_nums2[i] <- x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),][,2]}
      else if (nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),]) == 0 |
               is.null(nrow(x$author_address[(x$author_address$ut == papers[y,2]) &
(x$author_address$author_no == author_no),])))
      {print("Making resarchers only address NA, as nrows for author not == 0 |
is.null(researchers address no.)")
        addy_nums2[i] <- NA }
    }
    #Replace state numbers array with updated state number array only taking first address
for each
    #author as default to correct unequal rows in get papers when extracting address data.
    Addy_nums <- as.array(unlist(addy_nums2))
    print("New addy numbers are")
    print(Addy_nums)
  }
  for (i in 1:length(Addy_nums))
  {
    print(Addy_nums[i])
    if (length(Addy_nums) == 0 )
    {Addy_nums[i] <- NA}

    else {Addy <- x$address[x$address$ut == papers[y,2] & x$address$addr_no ==
Addy_nums[i], ADDRESS_COL]
    print(paste("here",Addy))
    if (length(Addy) == 0)
    {Addy_nums[i] = NA}
    else
    {Addy_nums[i] = Addy}}

  }
  return(Addy_nums)}

  else {print("end")
    return(NA)}}

#Creating main WOS extraction function:
get_papers <- function(x){
  #Extracting all authors papers with DOI, ut (WOS -specific paper identifier) & titles:
  papers <- data.frame(title = x$publication$title,
                       ut = x$publication$ut,
                       doi   = x$publication$doi)
```

```
  if (!nrow(papers)==0){
    paper.doi <- lapply(1:nrow(papers), function(y){
      if(length(papers!=0)){
        if(!is.na(papers[y,3]) & !(papers[y,3] == ''))
        {return(list(as.character(x$publication$doi[y])))}
        else if ((is.na(papers[y,3])|(papers[y,3] == '')) & (!is.na(papers[y,2]) &
!(papers[y,2] == '') ))
        {return(list(as.character(papers[y,2])))}
        else
        {return(NA)}}})
      your.papers <- lapply(1:nrow(papers), function(y){
        if(nrow(papers[y,]) == 0 | is.na(papers[y,])){
          data.frame(doi=NA,daisng_id=NA, firstname=NA,surname=NA
,ut=NA,abstract=NA,title=NA,venue=NA,date=NA,affil=NA,city=NA,state=NA,country=NA)
        } else {
          data.frame(doi = paper.doi[[y]][[1]]
                     ,daisng_id = x$author[x$author$ut==papers[y,2],7],
                     firstname = x$author[x$author$ut==papers[y,2],"first_name"],
                     surname = x$author[x$author$ut==papers[y,2],"last_name"],
                     ut = as.character(papers[y,2]),
                     abstract = x$publication[x$publication$ut == papers[y,2],7],
                     title = x$publication[x$publication$ut == papers[y,2],2],
                     venue = x$publication[x$publication$ut == papers[y,2],3],
                     date = str_split(as.character(x$publication[x$publication$ut ==
papers[y,2],4]), "-")[[1]][1],
                     affil = getaffil(y, x, papers),
                     city = getcity(y, x, papers),
                     state = getstate(y, x, papers),
                     country = getcountry(y, x, papers),
                     stringsAsFactors = FALSE)
        }})
      }
     else {your.papers <-
data.frame(doi=NA,daisng_id=NA,firstname=NA,surname=NA,ut=NA,abstract=NA,title=NA,venue=NA,
date=NA,affil=NA,city=NA,state=NA,country=NA) }
      do.call(rbind.data.frame, your.papers)
}
#Firstly, we will get the papers (get_papers) for all researchers that have UNIQUE
researcher/ORCID IDs from our WOS validation dataset vector.
print(paste("There are", length(WOS_vector), " researchers with scopus IDs - commencing
data extraction - possibly including duplicates"))

#Extracting co-authors of each researcher and concatenating into a list of co-authors and
other elements for each primary researcher - this will thereby get all first-order networks
of each researcher.
all.coauthors_first_order <- list()
 for (i in 1:length(WOS_vector))
{ cat(i, "of", length(WOS_vector), "WOS=", WOS_vector[i], "\n")
  if ((WOS_vector[i] != '') & (!is.na(WOS_vector[i])) & !WOS_vector[i] == "\"\"")
    { ID <- WOS_vector[i]
    #Preparing W-O-S specific Researcher IDs - AI= Author Identifiers
```

```
      query1 <- paste0('AI=(\"',ID,'\")')
      test <- pull_wos(query1, sid = sid)
      if (nrow(test$publication) > 0)
      # Download data for Researcher ID query for get papers function
      {all.coauthors_first_order[i] <- list(get_papers(pull_wos(query1, sid = sid)))
      print("WOS Extraction Successful")
      Extraction_summary_df[i, ] <-
c(UTS_researchers$name_clean[i],length(unique(all.coauthors_first_order[i][[1]]$doi)),"SUCC
ESSFUL","NOT-REQUIRED","SUCCESSFUL")}
      else if (nrow(test$publication) == 0)
      {
        print("WOS Researcher ID Extraction was unsuccessful - defaulting to ORCID
Extraction")
        ID <- ORCID_vector[i]
        if ((ID != '') & (!is.na(ID)))
        {
                if (nchar(ID) == 19)

                  {
                query1 <- paste0('AI=(\"',ID,'\")')
                test <- pull_wos(query1, sid = sid)
                if (nrow(test$publication) > 0)
                  # Download data for Researcher ID query for get papers function
                    {all.coauthors_first_order[i] <- list(get_papers(pull_wos(query1, sid
= sid)))

                    print("ORCID Extraction Successful")
                    Extraction_summary_df[i, ] <-
c(UTS_researchers$name_clean[i],length(unique(all.coauthors_first_order[i][[1]]$doi)),"UNSU
CCESSFUL","SUCCESSFUL","SUCCESSFUL")}
                else if (nrow(test$publication) == 0)
                {print(paste("ORCID ID extraction unsuccessful making", i, "NA
dataframe"))
                 all.coauthors_first_order[i] <-
list(data.frame(doi=NA,daisng_id=NA,firstname=NA,surname=NA,ut=NA,abstract=NA,title=NA,venu
e=NA,date=NA,affil=NA,city=NA,state=NA,country=NA))}
                Extraction_summary_df[i, ] <-
c(UTS_researchers$name_clean[i],0,"UNSUCCESSFUL","UNSUCCESSFUL","UNSUCCESSFUL")
}
          else if (nchar(ID) != 19)
          {print(paste(ID, "is not a ORCID identifer with length of", length(ID), "thus
ORCID extraction unsuccessful - defaulting to NULL"))
            all.coauthors_first_order[i] <-
list(data.frame(doi=NA,daisng_id=NA,firstname=NA,surname=NA,ut=NA,abstract=NA,title=NA,venu
e=NA,date=NA,affil=NA,city=NA,state=NA,country=NA))
            Extraction_summary_df[i, ] <-
c(UTS_researchers$name_clean[i],0,"UNSUCCESSFUL","NOT-PROVIDED","UNSUCCESSFUL")}}
        }   }

  else if  ((ORCID_vector[i] != '') & (!is.na(ORCID_vector[i])) & (ORCID_vector[i]
!="\"\"") )
```

```
    {print(paste(WOS_vector[i], "is either NA or '', defaulting to ORCID identifer:, ",
ORCID_vector[i]))
       ID <- ORCID_vector[i]
       if (nchar(ID) == 19)

      { query1 <- paste0('AI=(\"',ID,'\")')
        test <- pull_wos(query1, sid = sid)
        if (nrow(test$publication) > 0)
          # Download data for Researcher ID query for get papers function
        {all.coauthors_first_order[i] <- list(get_papers(test))
        print("ORCID Extraction Successful")
        Extraction_summary_df[i, ] <-
c(UTS_researchers$name_clean[i],length(unique(all.coauthors_first_order[i][[1]]$doi)),"NOT-
PROVIDED","SUCCESSFUL","SUCCESSFUL")}
        else if (nrow(test$publication) == 0)
        {print(paste("ORCID ID extraction unsuccessful making", i, "NA dataframe"))
         all.coauthors_first_order[i] <-
list(data.frame(doi=NA,daisng_id=NA,firstname=NA,surname=NA,ut=NA,abstract=NA,title=NA,venu
e=NA,date=NA,affil=NA,city=NA,state=NA,country=NA))
         Extraction_summary_df[i, ] <- c(UTS_researchers$name_clean[i],0,"NOT-
PROVIDED","UNSUCCESSFUL","UNSUCCESSFUL")}
      }
      else if (nchar(ID) != 19)
      {print(paste(ID, "is not a ORCID identifer with length of", length(ID), "thus ORCID
extraction unsuccessful - defaulting to NULL"))
        all.coauthors_first_order[i] <-
list(data.frame(doi=NA,daisng_id=NA,firstname=NA,surname=NA,ut=NA,abstract=NA,title=NA,venu
e=NA,date=NA,affil=NA,city=NA,state=NA,country=NA))
        Extraction_summary_df[i, ] <- c(UTS_researchers$name_clean[i],0,"NOT-
PROVIDED","NOT-PROVIDED","UNSUCCESSFUL")}}
    else if  ( ((WOS_vector[i] == '') | (is.na(WOS_vector[i]))) & ((ORCID_vector[i] == '')
| (ORCID_vector[i] == "\"\"") | (is.na(ORCID_vector[i]))) )
    {
      print(paste(WOS_vector[i], " WOS ID is NA or '' - defaulting to NA and",
ORCID_vector[i], "ORCID ID is also NA or ''- making NA dataframe"))
      all.coauthors_first_order[i] <-
list(data.frame(doi=NA,daisng_id=NA,firstname=NA,surname=NA,ut=NA,abstract=NA,title=NA,venu
e=NA,date=NA,affil=NA,city=NA,state=NA,country=NA))
      Extraction_summary_df[i, ] <- c(UTS_researchers$name_clean[i],0,"NOT-PROVIDED","NOT-
PROVIDED","UNSUCCESSFUL")
    }  }
#Moving to correct directory:
dir <- "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data"
setwd(dir)
#Saving WOS Extraction Summary and saving extracted WOS dataset for each primary
researcher:
Extraction_summary_df_WOS <- Extraction_summary_df
save(Extraction_summary_df_WOS, file = "Extraction_summary_df_WOS_v1.Rdata")
all.coauthors_first_order_WOS_EVAL_v1 <- all.coauthors_first_order
```

```
save(all.coauthors_first_order_WOS_EVAL_v1, file =
"all.coauthors_first_order_WOS_EVAL_v1.Rdata")
```

Script 1.4 Merging Three Extracted Rdata files for each respective pipeline into Merged Ambiguous Researcher Dataset

```
library(stringr)
library(xlsx)

#Creating final Extraction Summary document and saving output as excel:
load(file = "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data\\Extraction_summary_df_SCOPUS_v1.Rdata")
load(file = "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data\\Extraction_summary_df_PUBMED_v1.Rdata")
load(file = "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data\\Extraction_summary_df_WOS_v1.Rdata")
final_extraction_summary <- cbind(Extraction_summary_df_WOS,
Extraction_summary_df_PUBMED[,2:length(Extraction_summary_df_PUBMED)])
Extraction_summary_df_SCOPUS[11,] <- NA
final_extraction_summary <-
cbind(final_extraction_summary,Extraction_summary_df_SCOPUS[,2:length(Extraction_summary_df
_SCOPUS)])
save(final_extraction_summary, file = "C:\\Users\\Liam
Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data\\Extraction_summary_df_FINAL.Rdata")
write.xlsx(final_extraction_summary, "C:\\Users\\Liam
Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-Extraction\\Extraction_Summary.xlsx")
#Merging three networks together for stage two - disambiguation.
load(file = "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data\\all.coauthors_first_order_PUBMED_EVAL_v1.Rdata")
load(file = "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data\\all.coauthors_first_order_SCOPUS_EVALv1.Rdata")
load(file = "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data\\all.coauthors_first_order_WOS_EVAL_v1.Rdata")
all.coauthors_first_order_PUBMED_EVAL_v1 <- all.coauthors_first_order_EVAL_v1
#Converting these from list to dataframes:
all.coauthors_first_order_PUBMED_EVAL_v1 <-
do.call(rbind.data.frame,all.coauthors_first_order_PUBMED_EVAL_v1)
all.coauthors_first_order_SCOPUS_EVALv1 <-
do.call(rbind.data.frame,all.coauthors_first_order_SCOPUS_EVALv1)
all.coauthors_first_order_WOS_EVAL_v1 <-
do.call(rbind.data.frame,all.coauthors_first_order_WOS_EVAL_v1)
#Taking variables used from disambig paper for merging together in stage two. *Can be
updated if additional variables used (e.g. country, citations or publication date)
PUBMED_df <- all.coauthors_first_order_PUBMED_EVAL_v1[, c("surname", "firstname", "venue",
"affil", "title", "abstract", "doi","year")]
#Cleaning date to year-only for WOS:
for (i in 1:nrow(all.coauthors_first_order_WOS_EVAL_v1)){
  all.coauthors_first_order_WOS_EVAL_v1$date_clean[i] <-
str_split(as.character(all.coauthors_first_order_WOS_EVAL_v1$date[i]),"-")[[1]][1]
}
all.coauthors_first_order_WOS_EVAL_v1$year <-
all.coauthors_first_order_WOS_EVAL_v1$date_clean
```

```
WOS_df <- all.coauthors_first_order_WOS_EVAL_v1[, c("surname", "firstname", "venue",
"affil", "title", "abstract", "doi","year")]
all.coauthors_first_order_SCOPUS_EVALv1$venue <-
all.coauthors_first_order_SCOPUS_EVALv1$Venue
SCOPUS_df <- all.coauthors_first_order_SCOPUS_EVALv1[, c("surname", "firstname", "venue",
"affil", "title", "abstract", "doi","year")]
final_df <- rbind(WOS_df, SCOPUS_df)
final_df <- rbind(final_df,PUBMED_df)
final_df <- unique(final_df)
final_df <- final_df[!is.na(final_df$surname),]
#Cleaning Primary and Secondary Researcher names:
name_clean <- function(i)
{
  surname_comps <- strsplit(toupper(str_trim(final_df$surname[i])), " |-|'|,")
  surname <- paste(unlist(surname_comps), collapse="")
  first <- substr(toupper(str_trim(final_df$firstname[i])), 1, 1)
  final <- paste(surname, first,sep=",")
  return(final)}
for (i in 1:nrow(final_df))
{  final_df$name_clean[i] <- name_clean(i)}
#Saving Merged Primary and Secondary Researcher Dataset:
save(final_df,file="C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data\\all.coauthors_final_df.Rdata")
```

Note: To be fully functional and generalizable for user's directory variables would need to be dynamic rather than fixed for original pathway in creators' environment.

9.2-3 Researcher Profile Disambiguation & Consolidation

The Disambiguation & Consolidation of stage two to three, this involves a mix of Python and R Scripts, loading in Rdata files and converting these to XML files for the Python disambiguation algorithm in R and then running in this algorithm in python. Future iterations would remove ORCID publications being added as an assumption of being correct for all researchers at end of the disambiguation and instead place this ORCID publication profile extraction in stage one extraction to be included in the disambiguation process and thus reduce false positives. Secondly, for using reticulate (in joining R and Python scripts into a single application in Rshiny) this algorithm must be uphauled from Python 2 (NetworkX version 1 to NetworkX version 2). A key update for this section to improve performance would be the inclusion of publication year, country, citation network and additional dimensional layers to increase performance and thereby add layers in the publication network constructions – these data points are already extracted in stage one, though not currently utilised.

Script 2-3.1 Conversion of Merged Rdata Extracted Ambiguous Researcher Profiles to XML files for Disambiguation

---

```
#Importing Necessary Libraries:
library(XML)
library(xml2)
library(stringr)
library(rscopus)
library(rorcid)

#Moving to correct directory:
dir <- "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\University_Input_datasets"
setwd(dir)

#Loading in the SPHERE Researchers Dataset:
UTS_researchers <- read.csv("TEST_10_SAMPLE_data_RESEARCHERS.csv",
                            header = TRUE,
                            quote="\"",
                            stringsAsFactors= TRUE,
                            strip.white = TRUE)

#Loading in Validation Dataset:
UTS_PUBLICATIONS <- read.csv("TEST_10_SAMPLE_data_PUBS_MASTER.csv",
                             header = TRUE,
                             quote="\"",
                             stringsAsFactors= TRUE,
                             strip.white = TRUE)

#Removing duplicate researcher records (due to more than 1 row for researchers with > 1
IDs) as this creates bug in extraction.
UTS_researchers <- UTS_researchers[UTS_researchers$ORCID.ID != "",]
#cleaning for publications validation:
for (row in 1:nrow(UTS_PUBLICATIONS))
{
  if (UTS_PUBLICATIONS$RE_flag[row] != "no"){
    re_key = UTS_PUBLICATIONS$RE_flag[row]
    print(re_key)
```

```
    #UTS_PUBLICATIONS$ALL_AUTHORS[row] <-
str_replace(as.character(UTS_PUBLICATIONS$ALL_AUTHORS[row]), "-|'","")
    test <- str_trim(str_replace(as.character(UTS_PUBLICATIONS$ALL_AUTHORS[row]), "-
|'|\\*",""))
    authors = str_extract_all(as.character(test), "\\w+, \\w+")
    print(authors)
  }
  if (UTS_PUBLICATIONS$RE_flag[row] == "no"){
    test <- str_trim(str_replace(as.character(UTS_PUBLICATIONS$ALL_AUTHORS[row]), "-
|'|\\*",""))
    between_split_key = as.character(UTS_PUBLICATIONS$ALL_COAUTHORS_KEY[row])
    authors = str_split(as.character(test), between_split_key)
    print(authors)
  }
  within_AUTH_key = as.character(UTS_PUBLICATIONS$within_AUTH_key[row])
  print(within_AUTH_key)
  if (within_AUTH_key == "" | within_AUTH_key == "\"\"" ){
    within_AUTH_key = " "
  }
  name_schema <- as.character(UTS_PUBLICATIONS$name_schema[row])
  list_index <- 0
  cleaned_author_list <- list()
  cleaned_author_list[1] = ""
  for (i in 1:length(authors[[1]])){



    name = str_split(authors[[1]][i], as.character(within_AUTH_key))
    index <- 0
    cleaned_name <- list()
    cleaned_name[[1]] = " "
    for (j in 1:length(name[[1]])){
      if (name[[1]][j] != ""){
        index <- index + 1
        cleaned_name[[1]][index] <- name[[1]][j]
      }
    }
    name <-  cleaned_name
    if (name_schema == "last,first"){
      lname <- name[[1]][1]
      fname <-  substr(name[[1]][length(name[[1]])],1,1)
      cleaned <- paste(str_to_upper(str_trim(lname)),str_to_upper(str_trim(fname)),
collapse=",", sep=",")
    }
    if (name_schema == "first,last")
    { lname <- name[[1]][length(name[[1]])]
    fname <- substr(name[[1]][1],1,1)
    cleaned <- paste(str_to_upper(str_trim(lname)),str_to_upper(str_trim(fname)),
collapse=",", sep=",")
    }
    list_index <- list_index + 1
```

```
    cleaned_author_list[[1]][list_index] = cleaned
  }
  cleaned_author_list <- paste(unlist(cleaned_author_list), collapse="; ")
  UTS_PUBLICATIONS$cleaned_author_list[row] <- cleaned_author_list
}

#cleaning for publications valdiation:
x <- str_split(UTS_PUBLICATIONS$cleaned_author_list, ";")

UTS_PUBLICATIONS$ALL_AUTHORS_CLEANED <- sapply(1:length(x), function(i) {
  name_cleaned <- sapply(1:length(x[[i]]), function(j){
    name <- str_split(str_trim(x[[i]][j])," |-|'|,")
    if (length(name[[1]]) <= 2){
      lname <- name[[1]][1]
      if (str_trim(lname) == ",") {
        lname <- "NA"
      }
      fname <- str_sub(name[[1]][2],1,1)
      if (str_trim(fname) == ",") {
        fname <- "NA"
      }

      }
    if (length(name[[1]]) > 2){
      lname <- paste(name[[1]][1:length(name[[1]])-1],sep="",collapse="")
      if (str_trim(lname) == ",") {
        lname <- "NA"
      }
      fname <- str_sub(name[[1]][length(name[[1]])],1,1)
      if (str_trim(fname) == ",") {
        fname <- "NA"
      }    }
    cleaned <-   str_to_upper(paste(lname,fname,sep=","))
    return(cleaned)})
  return(unlist(name_cleaned))})
UTS_PUBLICATIONS <- UTS_PUBLICATIONS[,c("ALL_AUTHORS_CLEANED",
"ARTICLE_CHAPTER_PAPER_TITLE", "JOURNAL_TITLE","YEAR")]


#Function for ORCID profile merging with reference networks for all of primary researchers
getabstract <- function(x){
  abstract = abstract_retrieval(papers[x,2], identifier = "doi")
  if (length(abstract$content) != 0){
    if (length(abstract$content$`abstracts-retrieval-response`) != 0){
  if (length(abstract$content$`abstracts-retrieval-response`$item$bibrecord$head$abstracts)
!= 0)
  {return(abstract$content$`abstracts-retrieval-response`$item$bibrecord$head$abstracts)}}}
  else {return(NA)}}

get_doi <- function(x){
  #  This pulls the DOIs out of the ORCiD record:
```

```
  list.x <- x$`external-ids.external-id`
  #  Catching objects with NULL DOI information:
  do.call(rbind.data.frame,lapply(list.x, function(x){

    if(length(x)==0){data.frame(value=NA)}
    else
    {
      if(((!'doi' %in% x[,1]) & (!'DOI' %in% x[,1]) )){
        data.frame(value=NA)
      } else{
        data.frame(value = x[which(x[,1] %in% 'doi'|x[,1] %in% 'DOI'),2])
      }}  }))  }
#Again, using get_papers function for retrieving ORCID researcher papers from ORCID API as
fourth publication pipeline for both researcher reference profiles AND main researcher
profiles later in file
get_papers <- function(x){
  all.papers <- x[[1]][[1]] # this is where the papers are for ORCID API
  org_temp <- NA
  try( org_temp <- orcid_employments(UTS_researchers$ORCID.ID[j])[[1]]$`affiliation-
group`$summaries[[1]][23])
  if (is.null(org_temp)) {org_temp <- NA}
  print(paste('Org temp (organisation) is', org_temp))
  jconf <- as.character(all.papers$`journal-title.value`)
  if (is.null(jconf)) {jconf <- NA}

  papers <- data.frame(title = all.papers$'title.title.value',
                       doi   = get_doi(all.papers),
                       org = org_temp,
                       jconf = jconf,
                       date = all.papers$`publication-date.year.value`

  )


  paper.doi <- lapply(1:nrow(papers), function(h){
    if(length(papers!=0)){
      if(!is.na(papers[h,2]))
      {return(orcid_doi(dois = check_dois(papers[h,2])$good, fuzzy = FALSE))}
      else
      {return(NA)}}})

  your.papers <- lapply(1:length(paper.doi), function(x){
    if(length(paper.doi[[x]]) == 0 || is.na(paper.doi[[x]])){
     #data.frame(doi=NA, orcid=NA, name_clean=NA,
organisation=NA,jconf=NA,date=NA,abstract=NA, title = NA)
      data.frame(doi=NA, orcid=NA, name_clean=NA, organisation=NA,jconf=NA,date=NA, title =
NA)
    } else {
      name_vec <- c()
      name_space <- orcid_person(paper.doi[[x]][[1]]$'orcid-identifier.path')
      if (length(name_space)>0){
```

```
        for (j in 1:length(name_space))
        {
          if (length(name_space[[j]]$name$`given-names`$value)==0 &
length(name_space[[j]]$name$`family-name`$value==0)){
            name_vec <- NA}
          else if(length(name_space[[j]]$name$`given-names`$value)>0 &
length(name_space[[j]]$name$`family-name`$value>0))
          {
            name_vec[j] <- paste(name_space[[j]]$name$`given-names`$value,
name_space[[j]]$name$`family-name`$value, sep=' ')
          }

          else if ((is.null(name_space[[j]]$name$`given-names`$value) &
is.null(name_space[[j]]$name$`family-name`$value)))
          {#print("NAME SPACE IS NULL!!!!!!!!!!!!!!!!!!!!!!!!!!")
            name_vec <- NA}
          else {print("ELSE EXCEPTION - CONDITION NOT CAUGHT FOR NAME_SPACE")
            print(paste("VALUE IS:",name_space[[j]]$name$`given-names`$value,
":name_space[[j]]$name$`given-names`$value - FINAL"))
            print(paste("VALUE IS:",name_space[[j]]$name$`family-name`$value,
"name_space[[j]]$name$`family-names`$value - FINAL"))}

        }}
      data.frame(doi = papers[x,2],
                 orcid = paper.doi[[x]][[1]]$'orcid-identifier.path',
                 name_clean = name_vec,
                 organisation = papers[x,3],
                 jconf = papers[x,4],
                 date = papers[x,5],
                 title = papers[x,1],
                 #abstract = getabstract(x),
                 stringsAsFactors = FALSE)
    })
  do.call(rbind.data.frame, your.papers)
}

#Moving to correct directory:
dir <- "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\2-3 -
Disambiguation and Network Comparison (IGM)\\2-Running Disambiugation and Cluster
Comparison"
setwd(dir)
UTS_researchers$name_clean <- sapply(1:nrow(UTS_researchers),function(y){
  lname <- str_to_upper(str_trim(UTS_researchers$SURNAME[y]))
  lname_clean <- str_split(lname, " |-|'")
  lname_clean <- paste(lname_clean[[1]], sep="",collapse="")
  print(lname_clean)
  fname <-str_sub(str_to_upper(str_trim(UTS_researchers$FIRST_NAME[y])),1,1)
  print(fname)
  return(paste(lname_clean,fname,sep=",",collapse=","))
})
unique_authors <- unique(UTS_researchers$name_clean)
```

```
#Extracting authors papers for each university author:
for (i in 1:length(unique_authors)){
  pub_title_list <- list()
  pub_count <- 1
  ego <- unique_authors[i]
  print(paste(ego, "ego"))
  for (j in 1:nrow(UTS_PUBLICATIONS))
  {
    pub_author_list <- UTS_PUBLICATIONS$ALL_AUTHORS_CLEANED[[j]]
    #print(pub_author_list)
    if (ego %in% pub_author_list &
!(as.character(UTS_PUBLICATIONS$ARTICLE_CHAPTER_PAPER_TITLE[j]) %in% pub_title_list)) {
      pub_title_list[pub_count] <-
as.character(UTS_PUBLICATIONS$ARTICLE_CHAPTER_PAPER_TITLE[j])
      print(pub_count)
      pub_count <- pub_count + 1
      UTS_PUBLICATIONS$ALL_AUTHORS_CLEANED[j]
    }}
  if (length(pub_title_list)>0){

    UTS_researchers$AUTHOR_PUBS[i] <- paste(unlist(pub_title_list),sep="<>", collapse="<>")
    UTS_researchers$pub_count[i] <- pub_count-1
  }
  else {print(paste(pub_title_list, "not accepted possibly empty (no pubs for author,
trying ORCID extraction for author", ego))
    UTS_researchers$AUTHOR_PUBS[i] <- NA
    UTS_researchers$pub_count[i] <- pub_count-1}
  print("AUTHOR HAS NO PAPERS!!!!!!!!!!!!!!!!!!")
}
UTS_PUBLICATIONS$Venue <- as.character(UTS_PUBLICATIONS$JOURNAL_TITLE)



#Creating total publication count dictionary for main and refs for primary authors:
total_cnt_df_ref <- data.frame("author" <- NA,"ref_total_count"<- NA)
colnames(total_cnt_df_ref)[1] <- "author"
colnames(total_cnt_df_ref)[2] <- "ref_total_count"
rownames(total_cnt_df_ref) <- c()
```

```
BEGINNING OF PRIMARY_REFERENCE XML FILE PREPARATION
#Creating the abstract_title.txt for author disambiguation algorithm for primary.
#Get primary ref network XML files ready.
#Check to see that ref network pipeline is the same for main pipeline.
#if ORCID ID then attempt ORCID extraction for supplement to reference network:
  #only if name matching between UNI database and ORCID DBS.
Extraction_summary_df <- data.frame(Name=NA, ORCID_REF_Total=NA, OVERALL_ORCID_STATUS=NA)
for (j in 1:nrow(UTS_researchers)){
  ego <- UTS_researchers$name_clean[j]
  title_doi_list <- list()
  doi_count <- 1
```

```
  pub_cnt <- 0
  duplicate_count <- 0
  ego_pub_list <- unique(str_split(UTS_researchers$AUTHOR_PUBS[j],"<>")[[1]])

  #Creating the XML raw file Tree format root for author
  prefix.xml <- "
  <author_set>
</author_set>
  "
  doc = xmlTreeParse(prefix.xml, useInternalNodes = T,addFinalizer = TRUE)
  root = xmlRoot(doc)  #FIND ROOT
  #firstname tag#
  FnameNode = newXMLNode("FullName",ego, parent=root)
  print("Beginning XML extraction for university dbs extraction datasets for researcher
reference network")
  for (i in 1:length(ego_pub_list)){
    dup <- FALSE
    title <- str_to_lower(str_trim(as.character(unique(ego_pub_list[i]))))
    #Cleaning author title:
    if (!is.na(title)){
      if (str_detect(substr(title, nchar(title), nchar(title)), "^[^\\p{L}\\p{Nd}]+$") ==
TRUE){
        print(paste("Cleaning  title:",substr(title, nchar(title), nchar(title)),"from
title end:",title))
        title <- substr(title,1,nchar(title)-1)
      }
      if (str_detect(substr(title, 1, 1), "^[^\\p{L}\\p{Nd}]+$") == TRUE){
        print(paste("Cleaning  title:",substr(title, 1, 1),"from title beginning:",title))
        title <- substr(title,2,nchar(title))
      }}
      title_doi_list[doi_count] <- title
      doi_count <- doi_count + 1
    }
    else if (title %in% title_doi_list & !is.na(title))
    {dup <- TRUE
    print(paste("title", title, "in title doi duplicate list: will ignore publication"))
     duplicate_count <- duplicate_count + 1
     pub_cnt <- pub_cnt +1
     }
    if (dup == FALSE & !is.na(title) & !title == "NA") {
    print(paste("dup indicator is false - ", dup," - beginning XML extraction"))
    #increment pub count
    pub_cnt <- pub_cnt +1

    all.coauthors <-
str_trim(paste(as.character(unique(UTS_PUBLICATIONS[UTS_PUBLICATIONS$ARTICLE_CHAPTER_PAPER_
TITLE==ego_pub_list[i], ]$ALL_AUTHORS_CLEANED)[[1]]),sep=" ",collapse=" "))
    venue <-
str_to_lower(str_trim(as.character(unique(UTS_PUBLICATIONS[UTS_PUBLICATIONS$ARTICLE_CHAPTER
_PAPER_TITLE==ego_pub_list[i], ]$Venue)[[1]])))
```

```
    year <-
str_to_lower(str_trim(as.character(unique(UTS_PUBLICATIONS[UTS_PUBLICATIONS$ARTICLE_CHAPTER
_PAPER_TITLE==ego_pub_list[i], ]$YEAR)[[1]])))
    if (length(str_split(year,"-")[[1]]) > 1){
      print(paste(year, "YEAR NEEDS FORMATTING!"))
      year <- str_split(year,"-")[[1]][1]
      print(paste("formatted year is",year))
    }
    #Then this is one of the authors publications - add to authors XML file:
    #count node
    cnt_name_var <- paste("cntNode", pub_cnt, sep = "")
    assign(cnt_name_var, newXMLNode("pub_count",pub_cnt,  parent=root))
    #Note: pub_count can be removed for faster effi
    #publication node
    pub_name_var1 <- paste("pubNode", pub_cnt, sep = "")
    temp <- assign(pub_name_var1, newXMLNode("publication", parent=root))
    #title tag
    titl_name_var <- paste("titleNode", pub_cnt, sep = "")
    assign(titl_name_var, newXMLNode("title",title,  parent=temp))
    #jconf tag
    jconf_name_var <- paste("jconfNode", pub_cnt, sep = "")
    assign(jconf_name_var, newXMLNode("jconf",venue,  parent=temp))
    year_name_var <- paste("yearNode", pub_cnt, sep = "")
    assign(year_name_var, newXMLNode("year",year,  parent=temp))
    #affil_name_var <- paste("affilNode", pub_cnt, sep = "")
    #assign(affil_name_var, newXMLNode("organization",organisation,  parent=temp))
    #abstract tag
    #abstract_name_var <- paste("abstract", pub_cnt, sep = "")
    #assign(abstract_name_var, newXMLNode("abstract",abstract,  parent=temp))
    #doi tag
    #DOI_name_var <- paste("DOI", pub_cnt, sep = "")
    #assign(DOI_name_var, newXMLNode("DOI",doi,  parent=temp))
    #authors tag
    authors_name_var <- paste("authors", pub_cnt, sep = "")
    assign(authors_name_var, newXMLNode("authors",all.coauthors,  parent=temp))
    }}
  #Beginning ORCID phase for adding to XML file for author if author has ORCID ID:

  if (!UTS_researchers$ORCID.ID[j] == "\"\"" & !UTS_researchers$ORCID.ID[j] == '' &
!is.na(UTS_researchers$ORCID.ID[j])) {
      print("Author has ORCID XML extraction for researcher reference network")
    print("Checking if author surname, first inital is the same as author ego")
    orcid_name <- as.orcid(as.character(UTS_researchers$ORCID.ID[j]))
    if (!is.na(orcid_name) & !is.null(orcid_name)) {
    orcid_sname <- str_to_upper(str_trim(orcid_name[[1]]$name$`family-name`))
    if (length(orcid_sname)!=0){
    orcid_sname <- paste(str_split(orcid_sname, " |-|'")[[1]], sep="",collapse="")}
    else {orcid_sname <- NA}
    orcid_fname <-str_sub(str_trim(str_to_upper(orcid_name[[1]]$name$`given-names`)),1,1)
    if (length(orcid_fname)!=0){
    full_orcid_name <- paste(orcid_sname,orcid_fname,sep=",",collapse=",")
```

```
    }
    else {orcid_fname <- NA}
    print(paste("!!!!!!!!!!ORCID PROFILE NAME IS",full_orcid_name, "EGO (UNIVERSITY CLEANED
NAME) IS",ego,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!" ))
    if (full_orcid_name == ego)
    {      print("Names are the same beginning ORCID extraction for author as supplement to
University reference network for disambiguation.")
      #SUBSET HERE FOR UNIQUE UNSW RESEARCHERS ORCIDs for data extraction of papers and
coauthors.
      x <- orcid_works(UTS_researchers$ORCID.ID[j])
      if (!nrow(x[[1]][[1]]) == 0)
      {all.coauthors_first_order <- get_papers(x)
       orcid_df <- do.call(rbind.data.frame,list(all.coauthors_first_order))
      orcid_papers_count <- length(unique(orcid_df$doi))
      Extraction_summary_df[j, ] <- c(ego,orcid_papers_count,"SUCCESSFUL")
      #Now cleaning the name_clean from extracted ORCID profile in orcid_df:
      for (y in 1:nrow(orcid_df)){
        if (!is.na(orcid_df$name_clean[y]) & !is.null(orcid_df$name_clean[y])) {
          name <- str_split(str_to_upper(str_trim(orcid_df[y,'name_clean'])),pattern=" ")
          orcid_fname <-str_sub(str_trim(str_to_upper(name[[1]][1])),1,1)
          sname <- name[[1]][2:length(name[[1]])]
          orcid_sname <- paste(str_split(sname, pattern=" |-|'")[[1]], sep="",collapse="")
          full_orcid_name <- paste(orcid_sname,orcid_fname,sep=",",collapse=",")
          orcid_df$name_clean[y] <- full_orcid_name
        }}
      print(paste("!!!!!!!!!!ORCID PROFILE NAME IS",full_orcid_name, "EGO (UNIVERSITY
CLEANED NAME) IS",ego,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!" ))
      if (!is.null(orcid_df))
      {
        pub_df <- data.frame("title" <- NA, "author_list" <- NA)
        colnames(pub_df)[1] <- "title"
        colnames(pub_df)[2] <- "author_list"
        ##View(pub_df)
        dataset_unique <- unique(orcid_df$title)
        #print(dataset_unique)
        #Creating unique pub dataframe with associated author_lists for each pub:
        for (z in 1:nrow(orcid_df))
        {
          #The publication to build author list for:
          pub <- dataset_unique[z]
          print(paste("pub is ",pub))
          if (!is.na(pub)){
          if (!pub %in% pub_df$title){
            author_list <- list("NULL")
            print(paste("Author_list is", author_list))
            cnt <- 0
            print(paste("Beginning count is:", cnt))
            for (i in 1:nrow(orcid_df)){

            if (!is.na(orcid_df[i,"title"]) & !is.na(orcid_df[i,"name_clean"])){
```

```
              if (orcid_df[i,"title"] == pub & !(orcid_df[i,"name_clean"] %in%
author_list)){
                author_list[cnt + 1] <- orcid_df[i,"name_clean"]
                print(paste("Author",orcid_df[i,"name_clean"], "not in author_list so
adding", unlist(orcid_df[i,"name_clean"]), "to", "author_list at index", cnt + 1 ))
                cnt <- cnt + 1
              }

            }
            pub_df[z,"title"] <- as.character(dataset_unique[z])
            pub_df[z,"author_list"] <- paste(author_list, collapse = " ")
            print(pub_df[z,])}}
        }}}
        #Now applying author_lists to each pubication of each author record
        for (y in 1:nrow(orcid_df))
        {   if (!is.na(orcid_df[y, "title"])){
           test <- pub_df[pub_df$title == orcid_df[y, "title"],c("author_list")]
           for (k in 1:length(test)){
             if (length(test) > 0){
               if (!is.na(test[[k]]))  {orcid_df[y, "author_list"] <- test[[k]]}

            }}         }}


      #XML extraction:

        for (y in 1:nrow(orcid_df)){
        if (!is.na(orcid_df[y,"name_clean"]) & !is.na(orcid_df[y,"title"])){
          if (orcid_df[y,"name_clean"] == ego){
            #Checking paper is not a duplicate:
            dup <- FALSE
            doi <- str_to_lower(str_trim(as.character(unique(orcid_df[y,"doi"]))))
            title <- str_to_lower(str_trim(as.character(unique(orcid_df[y,"title"]))))
            year <- str_trim(as.character(unique(orcid_df[y,"date"])))
            if (length(str_split(year,"-")[[1]]) > 1){
              print(paste(year, "YEAR NEEDS FORMATTING!"))
              year <- str_split(year,"-")[[1]][1]
              print(paste("formatted year is",year))
            }
            #Cleaning author title:
            if (!is.na(title)){
              if (str_detect(substr(title, nchar(title), nchar(title)),
"^[^\\p{L}\\p{Nd}]+$") == TRUE){
                print(paste("Cleaning orcid title:",substr(title, nchar(title),
nchar(title)),"from title end:",title))
                title <- substr(title,1,nchar(title)-1)
              }
              if (str_detect(substr(title, 1, 1), "^[^\\p{L}\\p{Nd}]+$") == TRUE){
                print(paste("Cleaning orcid title:",substr(title, 1, 1),"from title
beginning:",title))
                title <- substr(title,2,nchar(title))
```

```
            }}
          if (!doi %in% title_doi_list & !title %in% title_doi_list & !is.na(title)){
            print(paste("doi", doi, "and title", title, "not in title doi duplicate list
- adding to list and commencing extraction:"))
            if (!is.na(doi) & !doi == " ")
            {title_doi_list[doi_count] <- doi
            doi_count <- doi_count + 1}
            title_doi_list[doi_count] <- title
            doi_count <- doi_count + 1
          }
          else if (doi %in% title_doi_list | title %in% title_doi_list & !is.na(title))
          {dup <- TRUE
          print(paste("title", title,"or doi", doi, "in title doi duplicate list: will
ignore publication - ORCID EXTRACTION"))
          duplicate_count <- duplicate_count + 1
          pub_cnt <- pub_cnt +1}
          if (dup == FALSE & !is.na(title) & !title == "NA") {
            print(paste("dup indicator is false - ", dup," - beginning XML ORCID
extraction"))
          #increment pub count
          pub_cnt <- pub_cnt +1
          #Then this is one of the authors publications - add to authors XML file:
          #count node
          cnt_name_var <- paste("cntNode", pub_cnt, sep = "")
          assign(cnt_name_var, newXMLNode("pub_count",pub_cnt,  parent=root))
          #Note: pub_count can be removed for faster effi
          #publication node
          pub_name_var1 <- paste("pubNode", pub_cnt, sep = "")
          temp <- assign(pub_name_var1, newXMLNode("publication", parent=root))
          #title tag
          #cleaning orcid extracted title:
          title <- str_trim(str_to_lower(title))
          if (str_detect(substr(title, nchar(title), nchar(title)),
"^[^\\p{L}\\p{Nd}]+$") == TRUE){
            print(paste("CLEANED:",substr(title, nchar(title), nchar(title))))
            title <- substr(title,1,nchar(title)-1)
          }
          titl_name_var <- paste("titleNode", pub_cnt, sep = "")
          assign(titl_name_var, newXMLNode("title",title,  parent=temp))
          #jconf tag
          jconf <- str_to_lower(str_trim(orcid_df[y,"jconf"]))
          jconf_name_var <- paste("jconfNode", pub_cnt, sep = "")
          assign(jconf_name_var, newXMLNode("jconf",jconf,  parent=temp))
          #organization tag
          organisation <- str_to_lower(str_trim(orcid_df[y,"organisation"]))
          affil_name_var <- paste("affilNode", pub_cnt, sep = "")
          assign(affil_name_var, newXMLNode("organization",organisation,  parent=temp))

          #year tag
          year_name_var <- paste("yearNode", pub_cnt, sep = "")
          assign(year_name_var, newXMLNode("year",year,  parent=temp))
```

```
            #doi tag
            DOI_name_var <- paste("DOI", doi, sep = "")
            assign(DOI_name_var, newXMLNode("DOI",doi,  parent=temp))
            #authors tag
            author_list <- str_trim(orcid_df[y,"author_list"])
            authors_name_var <- paste("authors", pub_cnt, sep = "")
            assign(authors_name_var, newXMLNode("authors",author_list,  parent=temp))
          }}}}}
        #Name the XML Doc:
        print(paste("!Scraped", pub_cnt, "ORCID profile publications for author!", ego))}

      else if (nrow(x[[1]][[1]]) == 0) {
        orcid_papers_count <- 0
        Extraction_summary_df[j, ] <- c(ego,orcid_papers_count,"UNSUCCESSFUL")
      }
        print(paste("THERE WAS", duplicate_count, "duplicate publications blocked between
ORCID and University database XML extraction"))
      }
}}

#Name the XML Doc:
print(paste("Scraped", pub_cnt, "total publications for author", ego))
#adding total count node:
total_cnt_name_var <- paste("total_cnt", pub_cnt-duplicate_count, sep = "")
assign(total_cnt_name_var, newXMLNode("total_cnt",pub_cnt-duplicate_count,  parent=root))
#Saving to total_cnt_df for disambig filtering/checks:
total_cnt_df_ref[j,]$author <- ego
total_cnt_df_ref[j,]$ref_total_count <- (pub_cnt-duplicate_count)
text<-read_xml(saveXML(doc))

#Save XML file for author:
if ((pub_cnt-duplicate_count) >0){
write_xml(text, file = paste("C:\\Users\\Liam
Ephraims\\Desktop\\LiamDissertation\\Evaluation\\2-3 - Disambiguation and Network
Comparison (IGM)\\2-Running Disambiugation and Cluster
Comparison\\PRIMARY_AUTHORS_XML\\REF\\"
                             ,paste(paste(unlist(str_split(ego,",")), collapse="."),
"_PRIMARY_REF.xml",collapse="",sep=""),sep=""), option = "as_xml")
print(paste(paste(paste(unlist(str_split(ego,",")), collapse="."),
"_PRIMARY_REF.xml",collapse="",sep=""), "XML file saved"))}
}
```

END OF PRIMARY REFERENCE XML FILE PREPARATION

```
#Preparing text file for author:total count ref dict.
setwd("C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\2-3 -
Disambiguation and Network Comparison (IGM)\\2-Running Disambiugation and Cluster
Comparison\\PRIMARY_AUTHORS_XML")
author_total <- total_cnt_df_ref
author_total <- unique(author_total)
formatted_text <- ""
```

```
for (i in 1:nrow(author_total))
{
  formatted_text <-
paste(formatted_text,paste(paste(author_total[i,"author"],author_total[i,"ref_total_count"]
, sep="<>"), collapse="\n"), "\n")
  print(length(formatted_text))
}
formatted_text <- paste("\n",formatted_text, sep="\n",collapse="\n")
write.table(formatted_text, file = "author_total_ref.txt", sep = "\t",
            row.names = FALSE, col.names = FALSE)

#Adding in ORCID count of publications extracted to extraction summary doc:
load(file = "C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data\\Extraction_summary_df_FINAL.Rdata")
final_extraction_summary <- cbind(final_extraction_summary, Extraction_summary_df)
write.xlsx(final_extraction_summary, "C:\\Users\\Liam
Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-Extraction\\Extraction_Summary.xlsx")

#Defining cleaning name function:
name_clean <- function(i, dataset)
{
  surname_comps <- strsplit(toupper(str_trim(dataset$surname[i])), " |-|'")

  #print(final_df$surname[i])
  surname <- paste(unlist(surname_comps), collapse="")
  #print(paste(unlist(surname_comps), collapse=""))

  first <- substr(toupper(str_trim(dataset$firstname[i])), 1, 1)
  #print(first)
  final <- paste(surname, first,sep=",")
  print(final)
  return(final)
}
```

```
#CREATING MAIN RESEARCHER XMLS FROM RDATA FILES TO XML FOR PYTHON DISAMBIGUATION
#Creating function for turning a given primary/secondary author main to XML & secondary
author ref network to XML:
author_extracted_dataset_to_XML <- function(dataset, network_type, secondary_ref_type){
  primary_auth_count <- 0
  #Creating total publication count dictionary for main and refs for primary authors:
  total_cnt_df_main <- data.frame("author" <- NA,"main_total_count"<- NA)
  colnames(total_cnt_df_main)[1] <- "author"
  colnames(total_cnt_df_main)[2] <- "main_total_count"
  rownames(total_cnt_df_main) <- c()
# Processing final dataset for format of author disambiguation and XML processing.
pub_df <- data.frame("title" <- NA, "author_list" <- NA)
colnames(pub_df)[1] <- "title"
colnames(pub_df)[2] <- "author_list"
dataset_unique <- unique(dataset$title)
#Creating unique pub dataframe with associated author_lists for each pub:
for (j in 1:length(dataset_unique))
```

```
{
  #The publication to build author list for:
  pub <- dataset_unique[j]
  print(paste("pub is ",pub))
  if (!pub %in% pub_df$title){
    author_list <- list("NULL")
    print(paste("Author_list is", author_list))
    cnt <- 0
    print(paste("Beginning count is:", cnt))
    for (i in 1:nrow(dataset)){
        if (dataset[i,"title"] == pub & !(dataset[i,"name_clean"] %in% author_list)){
          author_list[cnt + 1] <- dataset[i,"name_clean"]
          print(paste("Author",dataset[i,"name_clean"], "not in author_list so adding",
unlist(dataset[i,"name_clean"]), "to", "author_list at index", cnt + 1 ))
          cnt <- cnt + 1
        }}
    pub_df[j,"title"] <- dataset_unique[j]
    pub_df[j,"author_list"] <- paste(author_list, collapse = " ")
    print(pub_df[j,])}
  }
#Now applying author_lists to each pubication of each author record
for (y in 1:nrow(dataset))
{
  dataset[y, "author_list"] <- pub_df[pub_df$title == dataset[y, "title"],c("author_list")]
}
#Setting Directory for XML files to be saved to:
if (network_type == "primary"){
  setwd("C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\2-3 -
Disambiguation and Network Comparison (IGM)\\2-Running Disambiugation and Cluster
Comparison\\PRIMARY_AUTHORS_XML")}
else if (network_type == "secondary_ref"|network_type == "secondary_main"){
  setwd("C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\2-3 -
Disambiguation and Network Comparison (IGM)\\2-Running Disambiugation and Cluster
Comparison\\SECONDARY_AUTHORS_XML")}
#Check correct dir:
print(paste("Working directory for saving XML files is:",getwd()))
#Now creating  dataframe for each author consisting of XML tags for each of authosr
publications
name_cleans_unique <- unique(dataset$name_clean)

for (x in 1:length(name_cleans_unique)){
  print(paste("Beginning author", x, "out of", length(name_cleans_unique), "authors"))
  ego <- name_cleans_unique[x]
  title_doi_list <- list()
  doi_count <- 1
  pub_cnt <- 0
  duplicate_count <- 0
  pub_cnt <- 0
  print(paste("Ambiguous Author is", ego))
  print(paste("There are",nrow(dataset[dataset$name_clean == ego,]), "publications co-
authored by author", ego))
```

```
  #Creating the XML raw file Tree format root for author
  prefix.xml <- "
  <author_set>
</author_set>
  "
  doc = xmlTreeParse(prefix.xml, useInternalNodes = T,addFinalizer = TRUE)
  root = xmlRoot(doc)  #FIND ROOT
  #firstname tag
  FnameNode = newXMLNode("FullName",ego, parent=root)
  if (ego %in% unique_authors & network_type=="primary"){
    for (y in 1:nrow(dataset)){
      if (dataset[y,"name_clean"] == ego){
        dup <- FALSE
        doi <- str_to_lower(str_trim(as.character(unique(dataset[y,"doi"]))))
        title <- str_to_lower(str_trim(as.character(unique(dataset[y,"title"]))))
        if (!is.na(title)){
          if (str_detect(substr(title, nchar(title), nchar(title)), "^[^\\p{L}\\p{Nd}]+$")
== TRUE){
            print(paste("Cleaning  title:",substr(title, nchar(title), nchar(title)),"from
title end:",title))
            title <- substr(title,1,nchar(title)-1)
          }
          if (str_detect(substr(title, 1, 1), "^[^\\p{L}\\p{Nd}]+$") == TRUE){
            print(paste("Cleaning  title:",substr(title, 1, 1),"from title
beginning:",title))
            title <- substr(title,2,nchar(title))
          }}

        if (!doi %in% title_doi_list & !title %in% title_doi_list & !is.na(title)){
          print(paste("doi", doi, "and title", title, "not in title doi duplicate list -
adding to list and commencing extraction:"))
          if (!is.na(doi) & !doi == " ")
          {title_doi_list[doi_count] <- doi
          doi_count <- doi_count + 1}
          title_doi_list[doi_count] <- title
          doi_count <- doi_count + 1
        }
        else if (doi %in% title_doi_list | title %in% title_doi_list & !is.na(title))
        {dup <- TRUE
        print(paste("title", title,"or doi", doi, "in title doi duplicate list: will ignore
publication"))
        duplicate_count <- duplicate_count + 1
        pub_cnt <- pub_cnt +1
        }
        if (dup == FALSE & !is.na(title) & !title == "NA") {
          print(paste("dup indicator is false - ", dup," - beginning XML extraction for
primary main"))
        #increment pub count
        pub_cnt <- pub_cnt +1
        #Then this is one of the authors publications - add to authors XML file:
        #count node
```

```
        cnt_name_var <- paste("cntNode", pub_cnt, sep = "")
        assign(cnt_name_var, newXMLNode("pub_count",pub_cnt,  parent=root))
        #Note: pub_count can be removed for faster effi
        #publication node
        pub_name_var1 <- paste("pubNode", pub_cnt, sep = "")
        temp <- assign(pub_name_var1, newXMLNode("publication", parent=root))
        #title tag
        titl_name_var <- paste("titleNode", pub_cnt, sep = "")
        assign(titl_name_var, newXMLNode("title",dataset[y,"title"],  parent=temp))
        #jconf tag
        jconf <- str_to_lower(str_trim(dataset[y,"venue"]))
        jconf_name_var <- paste("jconfNode", pub_cnt, sep = "")
        assign(jconf_name_var, newXMLNode("jconf",jconf,  parent=temp))
        #organization tag
        affil <- str_to_lower(str_trim(dataset[y,"affil"]))
        affil_name_var <- paste("affilNode", pub_cnt, sep = "")
        assign(affil_name_var, newXMLNode("organization",affil,  parent=temp))
        #abstract tag
        abstract <- str_to_lower(str_trim(dataset[y,"abstract"]))
        abstract_name_var <- paste("abstract", pub_cnt, sep = "")
        assign(abstract_name_var, newXMLNode("abstract",abstract,  parent=temp))
        #doi tag
        DOI_name_var <- paste("DOI", pub_cnt, sep = "")
        assign(DOI_name_var, newXMLNode("DOI",doi,  parent=temp))
        #year tag
        year <- str_to_lower(str_trim(dataset[y,"year"]))
        if (length(str_split(year,"-")[[1]]) > 1){
          print(paste(year, "YEAR NEEDS FORMATTING!"))
          year <- str_split(year,"-")[[1]][1]
          print(paste("formatted year is",year))
        }
        year_name_var <- paste("year", pub_cnt, sep = "")
        assign(year_name_var, newXMLNode("year",year,  parent=temp))
        #authors tag
        authors_name_var <- paste("authors", pub_cnt, sep = "")
        assign(authors_name_var, newXMLNode("authors",dataset[y,"author_list"],
parent=temp))
      }}}
    #Name the XML Doc:
    #adding total count node:
    total_cnt_name_var <- paste("total_cnt", pub_cnt-duplicate_count, sep = "")
    assign(total_cnt_name_var, newXMLNode("total_cnt",pub_cnt-duplicate_count,
parent=root))
    #Saving to total_cnt_df for disambig filtering/checks:
    print(paste("HERE", total_cnt_df_main$author[j], ego))
    print(paste(total_cnt_df_main$main_total_count[j],(pub_cnt-duplicate_count)))
    print(paste("EGO", ego, length(ego), class(ego)))
    primary_auth_count <- primary_auth_count + 1
    total_cnt_df_main[primary_auth_count,]$author <- ego
    total_cnt_df_main[primary_auth_count,]$main_total_count <- (pub_cnt-duplicate_count)
    }
```

```
  else if (startsWith(network_type, 'secondary') & !(ego %in% unique_authors)){
    for (y in 1:nrow(dataset)){
      if (dataset[y,"name_clean"] == ego){
        dup <- FALSE
        doi <- str_to_lower(str_trim(as.character(unique(dataset[y,"doi"]))))
        title <- str_to_lower(str_trim(as.character(unique(dataset[y,"title"]))))
        if (!doi %in% title_doi_list & !title %in% title_doi_list & !is.na(title)){
          print(paste("doi", doi, "and title", title, "not in title doi duplicate list -
adding to list and commencing extraction:"))
          if (!is.na(doi) & !doi == " ")
          {title_doi_list[doi_count] <- doi
          doi_count <- doi_count + 1}
          title_doi_list[doi_count] <- title
          doi_count <- doi_count + 1
        }
        else if (doi %in% title_doi_list | title %in% title_doi_list)
        {dup <- TRUE
        print(paste("title", title,"or doi", doi, "in title doi duplicate list: will ignore
publication"))
        duplicate_count <- duplicate_count + 1
        pub_cnt <- pub_cnt +1
        }
        if (dup == FALSE & !is.na(title) & !title == "NA") {
          print(paste("dup indicator is false - ", dup," - beginning XML extraction for
secondary main"))
          #increment pub count
          pub_cnt <- pub_cnt +1
          #Then this is one of the authors publications - add to authors XML file:
          #count node
          cnt_name_var <- paste("cntNode", pub_cnt, sep = "")
          assign(cnt_name_var, newXMLNode("pub_count",pub_cnt,  parent=root))
          #publication node
          pub_name_var1 <- paste("pubNode", pub_cnt, sep = "")
          temp <- assign(pub_name_var1, newXMLNode("publication", parent=root))
          #title tag
          titl_name_var <- paste("titleNode", pub_cnt, sep = "")
          assign(titl_name_var, newXMLNode("title",title,  parent=temp))
          #jconf tag
          jconf <- str_to_lower(str_trim(dataset[y,"venue"]))
          jconf_name_var <- paste("jconfNode", pub_cnt, sep = "")
          assign(jconf_name_var, newXMLNode("jconf",jconf,  parent=temp))
          #organization tag
          affil <- str_to_lower(str_trim(dataset[y,"affil"]))
          affil_name_var <- paste("affilNode", pub_cnt, sep = "")
          assign(affil_name_var, newXMLNode("organization",affil,  parent=temp))
          #abstract tag
          abstract <- str_to_lower(str_trim(dataset[y,"abstract"]))
          abstract_name_var <- paste("abstract", pub_cnt, sep = "")
          assign(abstract_name_var, newXMLNode("abstract",abstract,  parent=temp))
          #year tag
```

```
          year <- str_to_lower(str_trim(dataset[y,"year"]))
          if (length(str_split(year,"-")[[1]]) > 1){
            print(paste(year, "YEAR NEEDS FORMATTING!"))
            year <- str_split(year,"-")[[1]][1]
            print(paste("formatted year is",year))
          }
          year_name_var <- paste("year", pub_cnt, sep = "")
          assign(year_name_var, newXMLNode("year",year,  parent=temp))

          #authors tag
          authors_name_var <- paste("authors", pub_cnt, sep = "")
         assign(authors_name_var, newXMLNode("authors",dataset[y,"author_list"],
parent=temp))
         #doi tag
         DOI_name_var <- paste("DOI", pub_cnt, sep = "")
         assign(DOI_name_var, newXMLNode("DOI",doi,  parent=temp))
      }}}
    #Name the XML Doc:
    #adding total count node:
    total_cnt_name_var <- paste("total_cnt", pub_cnt-duplicate_count, sep = "")
    assign(total_cnt_name_var, newXMLNode("total_cnt",pub_cnt-duplicate_count,
parent=root))
    #Saving to total_cnt_df for disambig filtering/checks:
    print(paste("HERE", total_cnt_df_main$author[j], ego))
    print(paste(total_cnt_df_main$main_total_count[j],(pub_cnt-duplicate_count)))
    print(paste("EGO", ego, length(ego), class(ego)))
    primary_auth_count <- primary_auth_count + 1
    total_cnt_df_main[primary_auth_count,]$author <- ego
    total_cnt_df_main[primary_auth_count,]$main_total_count <- (pub_cnt-duplicate_count)
    }
    text<-read_xml(saveXML(doc))

  #Save XML file for author and check if primary or secondary author:
  if (ego %in% unique_authors & network_type == "primary" & (pub_cnt-duplicate_count)>0)
#unique_authors is primary UNI DBS extracted authors.
  {  ".xml",collapse="",sep=""))) - not correctly fornatted (does not return spacing
between tags)
    write_xml(text, file = paste("C:\\Users\\Liam
Ephraims\\Desktop\\LiamDissertation\\Evaluation\\2-3 - Disambiguation and Network
Comparison (IGM)\\2-Running Disambiugation and CLuster
Comparison\\PRIMARY_AUTHORS_XML\\MAIN\\"
                                ,paste(paste(unlist(str_split(ego,",")), collapse="."),
"_PRIMARY_MAIN.xml",collapse="",sep=""),sep=""), option = "as_xml")
    print(paste(paste(paste(unlist(str_split(ego,",")), collapse="."),
"_PRIMARY_MAIN.xml",collapse="",sep=""), "XML file saved"))}

  else if (network_type == "secondary_main" & !(ego %in% unique_authors) & (pub_cnt-
duplicate_count)>0) {
    write_xml(text, file = paste("C:\\Users\\Liam
Ephraims\\Desktop\\LiamDissertation\\Evaluation\\2-3 - Disambiguation and Network
```

```
Comparison (IGM)\\2-Running Disambiugation and Cluster
Comparison\\SECONDARY_AUTHORS_XML\\MAIN\\"
                                ,paste(paste(unlist(str_split(ego,",")), collapse="."),
"_SECONDARY_MAIN.xml",collapse="",sep=""),sep=""), option = "as_xml")
      print(paste(paste(paste(unlist(str_split(ego,",")), collapse="."),
"_SECONDARY_MAIN.xml",collapse="",sep=""), "XML file saved"))}

  else if (network_type == "secondary_ref" & !(ego %in% unique_authors)){

    if (secondary_ref_type == "scopus"){
      write_xml(text, file = paste("C:\\Users\\Liam
Ephraims\\Desktop\\LiamDissertation\\UTS_COHORT_VALIDATION
SET\\DISAMBIGUATION\\Authors_XML\\secondary_authors\\scopus_ref\\",paste(paste(unlist(str_s
plit(ego,",")), collapse="."), "_SECONDARY_REF_SCOPUS.xml",collapse="",sep=""),sep=""),
option = "as_xml")
      print(paste(paste(paste(unlist(str_split(ego,",")), collapse="."),
"_SECONDARY_REF_SCOPUS.xml",collapse="",sep=""), "XML file saved"))
    }
    else if (secondary_ref_type == "wos"){
      write_xml(text, file = paste("C:\\Users\\Liam
Ephraims\\Desktop\\LiamDissertation\\UTS_COHORT_VALIDATION
SET\\DISAMBIGUATION\\Authors_XML\\secondary_authors\\wos_ref\\",paste(paste(unlist(str_spli
t(ego,",")), collapse="."), "_SECONDARY_REF_WOS.xml",collapse="",sep=""),sep=""), option =
"as_xml")
      print(paste(paste(paste(unlist(str_split(ego,",")), collapse="."),
"_SECONDARY_REF_WOS.xml",collapse="",sep=""), "XML file saved"))
    }
    else if (secondary_ref_type == "pubmed"){
      write_xml(text, file = paste("C:\\Users\\Liam
Ephraims\\Desktop\\LiamDissertation\\UTS_COHORT_VALIDATION
SET\\DISAMBIGUATION\\Authors_XML\\secondary_authors\\pubmed_ref\\",paste(paste(unlist(str_s
plit(ego,",")), collapse="."), "_SECONDARY_REF_PUBMED.xml",collapse="",sep=""),sep=""),
option = "as_xml")
      print(paste(paste(paste(unlist(str_split(ego,",")), collapse="."),
"_SECONDARY_REF_PUBMED.xml",collapse="",sep=""), "XML file saved"))
    }
    else {print("NO secondary ref type defined")}
  }}
return(total_cnt_df_main)}

#Loading in Merged Ambiguous Researcher Publication Profiles from Rdata, Data Extraction,
stage one:
load(file="C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\1-
Extraction\\RExtracted_Researcher_data\\all.coauthors_final_df.Rdata")
final_df <- unique(final_df)
#Cleaning titles for de-duplication by removing trailing full stops and non-alphabetical
characters:
for (i in 1:length(final_df$title)){
  final_df$title[i] <- str_trim(str_to_lower(final_df$title[i]))
  if (str_detect(substr(final_df$title[i], nchar(final_df$title[i]),
nchar(final_df$title[i])), "^[^\\p{L}\\p{Nd}]+$") == TRUE){
```

```
    print(paste("CLEANED:",substr(final_df$title[i], nchar(final_df$title[i]),
nchar(final_df$title[i]))))
    final_df$title[i] <- substr(final_df$title[i],1,nchar(final_df$title[i])-1)
  }}
#Preparing secondary main XML files:
author_extracted_dataset_to_XML(dataset=final_df,network_type =
"secondary_main",secondary_ref_type="none")
#Preparing primary main XML files:
#TEST - final_df <- final_df[final_df$name_clean == "MEHTA,P",]
total_cnt_df_main = author_extracted_dataset_to_XML(dataset=final_df,network_type =
"primary",secondary_ref_type="none")

#Should now have primary main and reference and secondary mains and references for each
author of final dataset.
#Creating the author_totalcnt.txt for author disambiguation algorithm for primary authors
total counts
#Preparing text file for author:total count main dict.
author_total <- total_cnt_df_main
author_total <- unique(author_total)
formatted_text <- ""
for (i in 1:nrow(author_total))
{
  formatted_text <-
paste(formatted_text,paste(paste(author_total[i,"author"],author_total[i,"main_total_count"
], sep="<>"), collapse="\n"), "\n")
  print(length(formatted_text))
}
formatted_text <- paste("\n",formatted_text, sep="\n",collapse="\n")
write.table(formatted_text, file = "author_total_main.txt", sep = "\t",
            row.names = FALSE, col.names = FALSE)
```

```
#ORCID PUBLICATION ADDITION TO EXTRACTED RESEARCHER PROFILES:
#In later iterations of the research intelligence toolkit  the assumption of ORCID profiles
containing all correct researcher publications would be revoked, instead included ORCID as
a direct extraction pipeline in stage one extraction; thereby including ORCID publications
in the disambiguation process and hopefully significantly reducing false positive
occurrences within our stage one to three extraction, disambiguation and consolidation
process.
```

```
#Creating extracted primary author main ORCID profiles to add to disambiguated researcher
profiles after disambiguation:
#Creating the abstract_title.txt for author disambiguation algorithm for primary.
```

```
setwd("C:\\Users\\Liam Ephraims\\Desktop\\LiamDissertation\\Evaluation\\2-3 -
Disambiguation and Network Comparison (IGM)\\2-Running Disambiugation and Cluster
Comparison\\PRIMARY_AUTHORS_XML")
title_abstract <- final_df[, c("title", "abstract")]
title_abstract <- unique(title_abstract)
formatted_text <- ""
for (i in 1:nrow(title_abstract))
{
  formatted_text <- paste(formatted_text,paste(paste(title_abstract[i,"title"],
title_abstract[i,"abstract"], sep="<>"), collapse="\n"), "\n")
  print(length(formatted_text))
}
for (j in 1:length(unique_authors)){
  pub_cnt <- 0
  ego <- unique_authors[j]
  print(paste("Ambiguous Author is", ego))
  #Creating the XML raw file Tree format root for author
  prefix.xml <- "
  <author_set>
</author_set>
  "
  doc = xmlTreeParse(prefix.xml, useInternalNodes = T,addFinalizer = TRUE)
  help(xmlTreeParse)
  root = xmlRoot(doc)  #FIND ROOT
  #firstname tag#
  FnameNode = newXMLNode("FullName",ego, parent=root)
  #Adding in Authors ORCID papers if primary author:
  print("!Entering ORCIDID for MAIN!")
  orcid_id <- UTS_researchers[UTS_researchers$name_clean == ego &
!is.na(UTS_researchers$ORCID.ID)
                              & !UTS_researchers$ORCID.ID == "" & !UTS_researchers$ORCID.ID
== "\"\"" ,c("ORCID.ID")]
  print("ORCID ID IS:")
  print(orcid_id)
if (!length(orcid_id) == 0){
  if (!orcid_id == '' & !is.na(orcid_id) & !orcid_id == "\"\"") {
  print("Author has ORCID XML extraction for researcher main network")
  print("Checking if author surname, first inital is the same as author ego")
  orcid_name <- as.orcid(as.character(orcid_id))
  if (!is.na(orcid_name) & !is.null(orcid_name)) {
    orcid_sname <- str_to_upper(str_trim(orcid_name[[1]]$name$`family-name`))
    if (length(orcid_sname)!=0){
      orcid_sname <- paste(str_split(orcid_sname, " |-|'")[[1]], sep="",collapse="")}
    else {orcid_sname <- NA}
    orcid_fname <-str_sub(str_trim(str_to_upper(orcid_name[[1]]$name$`given-names`)),1,1)
    if (length(orcid_fname)!=0){
      full_orcid_name <- paste(orcid_sname,orcid_fname,sep=",",collapse=",")
    }
    else {orcid_fname <- NA}
```

```
    print(paste("!!!!!!!!!!ORCID PROFILE NAME IS",full_orcid_name, "EGO (UNIVERSITY CLEANED
NAME) IS",ego,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!" ))
    if (full_orcid_name == ego)
    {
      print("Names are the same beginning ORCID extraction for author as supplement to
University reference network for disambiguation.")
      #SUBSET HERE FOR UNIQUE UNSW RESEARCHERS ORCIDs for data extraction of papers and
coauthors.
      x <- orcid_works(orcid_id)
      print(x[[1]][[1]])
      if (!nrow(x[[1]][[1]]) == 0)
      {all.coauthors_first_order <- get_papers(x)
      orcid_df <- do.call(rbind.data.frame,list(all.coauthors_first_order))

      #Now cleaning the name_clean from extracted ORCID profile in orcid_df:
      for (y in 1:nrow(orcid_df)){
        if (!is.na(orcid_df$name_clean[y]) & !is.null(orcid_df$name_clean[y])) {
          name <- str_split(str_to_upper(str_trim(orcid_df[y,'name_clean'])),pattern=" ")
          orcid_fname <-str_sub(str_trim(str_to_upper(name[[1]][1])),1,1)
          sname <- name[[1]][2:length(name[[1]])]
          orcid_sname <- paste(str_split(sname, pattern=" |-|'")[[1]], sep="",collapse="")
          full_orcid_name <- paste(orcid_sname,orcid_fname,sep=",",collapse=",")
          orcid_df$name_clean[y] <- full_orcid_name
        }}
      print(paste("!!!!!!!!!!ORCID PROFILE NAME IS",full_orcid_name, "EGO (UNIVERSITY
CLEANED NAME) IS",ego,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!" ))
      if (!is.null(orcid_df))
      {
        pub_df <- data.frame("title" <- NA, "author_list" <- NA)
        colnames(pub_df)[1] <- "title"
        colnames(pub_df)[2] <- "author_list"
        ##View(pub_df)
        dataset_unique <- unique(orcid_df$title)
        #print(dataset_unique)
        #Creating unique pub dataframe with associated author_lists for each pub:
        for (z in 1:nrow(orcid_df))
        {
          #The publication to build author list for:
          #if (!is.na(dataset_unique[j])){
          # print(z)
          pub <- dataset_unique[z]
          print(paste("pub is ",pub))
          if (!is.na(pub)){
            if (!pub %in% pub_df$title){
              author_list <- list("NULL")
              print(paste("Author_list is", author_list))
              cnt <- 0
              print(paste("Beginning count is:", cnt))
              for (i in 1:nrow(orcid_df)){

                if (!is.na(orcid_df[i,"title"]) & !is.na(orcid_df[i,"name_clean"])){
```

```
                if (orcid_df[i,"title"] == pub & !(orcid_df[i,"name_clean"] %in%
author_list)){
                  author_list[cnt + 1] <- orcid_df[i,"name_clean"]
                  print(paste("Author",orcid_df[i,"name_clean"], "not in author_list so
adding", unlist(orcid_df[i,"name_clean"]), "to", "author_list at index", cnt + 1 ))
                  cnt <- cnt + 1
                }

              }
              pub_df[z,"title"] <- as.character(dataset_unique[z])
              pub_df[z,"author_list"] <- paste(author_list, collapse = " ")
              print(pub_df[z,])}

            #}
          }}}
      #Now applying author_lists to each pubication of each author record
      for (y in 1:nrow(orcid_df))
      {
        if (!is.na(orcid_df[y, "title"])){
          test <- pub_df[pub_df$title == orcid_df[y, "title"],c("author_list")]
          for (k in 1:length(test)){
            if (length(test) > 0){
              if (!is.na(test[[k]]))  {orcid_df[y, "author_list"] <- test[[k]]}

            }
          }
          #added first index look-up as second index returning NA
        }}
      #XML extraction:
      for (y in 1:nrow(orcid_df)){
        if (!is.na(orcid_df[y,"name_clean"]) & !is.na(orcid_df[y,"title"])){
          if (orcid_df[y,"name_clean"] == ego){
            #Checking paper is not a duplicate:
            dup <- FALSE
            doi <- str_to_lower(str_trim(as.character(unique(orcid_df[y,"doi"]))))
            title <- str_to_lower(str_trim(as.character(unique(orcid_df[y,"title"]))))
            year <- str_trim(as.character(unique(orcid_df[y,"date"])))
            if (length(str_split(year,"-")[[1]]) > 1){
              print(paste(year, "YEAR NEEDS FORMATTING!"))
              year <- str_split(year,"-")[[1]][1]
              print(paste("formatted year is",year))
            }
            #Cleaning author title:
            if (!is.na(title)){
              if (str_detect(substr(title, nchar(title), nchar(title)),
"^[^\\p{L}\\p{Nd}]+$") == TRUE){
                print(paste("Cleaning orcid title:",substr(title, nchar(title),
nchar(title)),"from title end:",title))
                title <- substr(title,1,nchar(title)-1)
              }
              if (str_detect(substr(title, 1, 1), "^[^\\p{L}\\p{Nd}]+$") == TRUE){
```

```
                print(paste("Cleaning orcid title:",substr(title, 1, 1),"from title
beginning:",title))
                title <- substr(title,2,nchar(title))
              }}

            #Checking if ORCID title is in the title-abstract dictionary if not then
attempting to add the abstract through SCOPUS:
            if (!title %in% title_abstract[,"title"]){
              abstract = abstract_retrieval(doi, identifier = "doi")
              if (length(abstract$content) != 0){
              if (length(abstract$content$`abstracts-retrieval-response`) != 0){
              if (length(abstract$content$`abstracts-retrieval-
response`$item$bibrecord$head$abstracts) != 0)
              { abstract = abstract$content$`abstracts-retrieval-
response`$item$bibrecord$head$abstracts}}}
              else { abstract = NA}
              if (!is.na(abstract)){
                rows <- length(title_abstract$title)
                title_abstract[rows + 1, "title"] <- title
                title_abstract[rows+ 1, "abstract"] <- abstract
                formatted_text <-
paste(formatted_text,paste(paste(title_abstract[rows+1,"title"],
title_abstract[rows+1,"abstract"], sep="<>"), collapse="\n"), "\n")
                print(length(formatted_text))
                print(paste("!ORCID TITLE AND ABSTRACT ADDED TO TITLE-ABSTRACT
DICTIONARY!"))
              }
}
            if (dup == FALSE & !is.na(title) & !title == "NA") {
              print(paste("dup indicator is false - ", dup," - beginning XML ORCID
extraction"))
              #increment pub count
              pub_cnt <- pub_cnt +1
              #Then this is one of the authors publications - add to authors XML file:
              #count node
              cnt_name_var <- paste("cntNode", pub_cnt, sep = "")
              assign(cnt_name_var, newXMLNode("pub_count",pub_cnt,  parent=root))
              #Note: pub_count can be removed for faster effi
              #publication node
              pub_name_var1 <- paste("pubNode", pub_cnt, sep = "")
              temp <- assign(pub_name_var1, newXMLNode("publication", parent=root))
              #title tag
              #cleaning orcid extracted title:
              title <- str_trim(str_to_lower(title))
              if (str_detect(substr(title, nchar(title), nchar(title)),
"^[^\\p{L}\\p{Nd}]+$") == TRUE){
                print(paste("CLEANED:",substr(title, nchar(title), nchar(title))))
                title <- substr(title,1,nchar(title)-1)
              }
              titl_name_var <- paste("titleNode", pub_cnt, sep = "")
              assign(titl_name_var, newXMLNode("title",title,  parent=temp))
```

```
                #jconf tag
                jconf <- str_to_lower(str_trim(orcid_df[y,"jconf"]))
                jconf_name_var <- paste("jconfNode", pub_cnt, sep = "")
                assign(jconf_name_var, newXMLNode("jconf",jconf,  parent=temp))
                #organization tag
                organisation <- str_to_lower(str_trim(orcid_df[y,"organisation"]))
                affil_name_var <- paste("affilNode", pub_cnt, sep = "")
                assign(affil_name_var, newXMLNode("organization",organisation,
parent=temp))
                #year tag
                year_name_var <- paste("yearNode", pub_cnt, sep = "")
                assign(year_name_var, newXMLNode("year",year,  parent=temp))
                #doi tag
                DOI_name_var <- paste("DOI", doi, sep = "")
                assign(DOI_name_var, newXMLNode("DOI",doi,  parent=temp))
                #authors tag
                author_list <- str_trim(orcid_df[y,"author_list"])
                authors_name_var <- paste("authors", pub_cnt, sep = "")
                assign(authors_name_var, newXMLNode("authors",author_list,  parent=temp))
              }}}}}
      #Name the XML Doc:
      print(paste("!Scraped", pub_cnt, "ORCID profile publications for author!", ego))}
      text<-read_xml(saveXML(doc))
      #Save XML file for author and check if primary or secondary author:
      if (ego %in% unique_authors & (pub_cnt)>0) #unique_authors is primary UNI DBS
extracted authors.
      {  #saveXML(doc, file= saveXML(doc, file=paste(paste(unlist(str_split(ego,",")),
collapse="."), ".xml",collapse="",sep=""))) - not correctly fornatted (does not return
spacing between tags)
        write_xml(text, file = paste("C:\\Users\\Liam
Ephraims\\Desktop\\LiamDissertation\\Evaluation\\2-3 - Disambiguation and Network
Comparison (IGM)\\2-Running Disambiugation and Cluster
Comparison\\PRIMARY_AUTHORS_XML\\ORCID\\"
                                    ,paste(paste(unlist(str_split(ego,",")),
collapse="."), "_PRIMARY_ORCID.xml",collapse="",sep=""),sep=""), option = "as_xml")
        print(paste(paste(paste(unlist(str_split(ego,",")), collapse="."),
"_PRIMARY_ORCID.xml",collapse="",sep=""), "XML file saved"))}
       }
  }}}
    }
#Creating final verison of title-abstract dict including latest ORCID titles.
formatted_text <- paste("\n",formatted_text, sep="\n",collapse="\n")
write.table(formatted_text, file = "title_abstract_disseration_test.txt", sep = "\t",
            row.names = FALSE, col.names = FALSE)
```

Script 2-3.2 Researcher Publication Profile Disambiguation & Consolidation converting Rdata to XML Researcher Profiles

Note: This file represents an implementation of network-based author disambiguation published by Xu et al (2018), we have implemented their code making minor adaptions including; adapting their pipeline for our researcher data, the creation of numerous check documents for reference and main profile counts, and removal of Affinity Propagation and amendment of their final output to include a Graph-Edit-Distance comparison stage three to disambiguated clusters, using this to take and compare a final cluster to represent as final disambiguated researcher for each primary researcher. Note: This implementation needs to be uphauled to Python 3 instead of Python 2 due to NetworkX, thereby allowing for integration of R scripts throughout project with Python scripts using R Reticulate, and allowing for creation of final Rshiny Application for GIU and functionality interface to users.

```
#Importing Necessary Libraries:
import os
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import numpy as np
import multiprocessing as mp
import pandas as pd
#NOTE: Need to install ged4py with - C:\WINDOWS\system32>pip install
git+https://github.com/Jacobe2169/ged4py.git#egg=ged4py
#To install latest version and fixes directly from Github.
from ged4py.algorithm import graph_edit_dist
from statistics import mean
import os.path
import networkx as nx
#Must be using networkx < VERSION 2:
#!pip uninstall networkx -y
#!pip install networkx==1.11
from jieba import analyse
#Creating reference profile publication count function:
def check_ref_count():
        ref_counts = open('PRIMARY_AUTHORS_XML/author_total_ref.txt','r')
        ref_counts_dict = {}
        for line in ref_counts.readlines():
            #Getting document of titles and abstracts divided by <>
            if "<>" in line:
                arr=line.split("<>")
                count = arr[1].strip().split(" \n")[0]
                #print(count)
                if len(arr)>1 and arr[1].strip() != "NA":
                    ref_counts_dict[arr[0].strip()]=int(count)
                else:
                    ref_counts_dict[arr[0].strip()]=0
        return ref_counts_dict
 #Creating main publication count function:
def check_main_count():
        main_counts = open('PRIMARY_AUTHORS_XML/author_total_main.txt','r')
        main_counts_dict = {}
```

```
        for line in main_counts.readlines():
            #Getting document of titles and abstracts divided by <>
            if "<>" in line:
                arr=line.split("<>")
                count = arr[1].strip().split(" \n")[0]
                if len(arr)>1 and arr[1].strip() != "NA":
                    main_counts_dict[arr[0].strip()]= int(count)
                else:
                    main_counts_dict[arr[0].strip()]=0
        return main_counts_dict
#Creating XML Reading to Disambiguated XML function:
def disambiguated_author_XMLconvertion(filename,file_path, x,ego,predict_label_dict_fin,
final_cluster_index):
    #Now saving cluster index papers for primary author to XML file:
    xmlFileName = file_path + "disambiguated_main/" + str(x)
    with open(filename, "r",encoding='utf-8') as filetoread, open(xmlFileName,
'w',encoding='utf-8') as xmlFile:
    # 这里是一行一行读取的文件
        print("Author ego is", ego, "extracting disambiguated author papers to XML:")
        disambiguated_author_profile = []
        xmlFile.write('<?xml version="1.0" encoding="UTF-8"?>\n')
        xmlFile.write('<author_set>\n')
        paper_index = 0
        n_extraction = 0
        title_doi_list = []
        doi_list = []
        title_list = []
        for line in filetoread:
            line = line.strip()
            #print("Stripped line",line)
            if "<publication>" in line:
                #INCREMENT PAPER COUNTER - index
                pub_start_XML = line
                paper_index += 1
                if paper_index -1 in predict_label_dict_fin[final_cluster_index]:
                    xmlFile.write(pub_start_XML + "\n")
                    n_extraction += 1
            elif "</publication>" in line and paper_index -1 in
predict_label_dict_fin[final_cluster_index]:
                pub_end_XML=line
                titlex = title_XML[title_XML.find('>')+1: title_XML.rfind('<')].strip()
                orgx = organization_XML[organization_XML.find('>')+1:
organization_XML.rfind('<')].strip()
                jconfx = jconf_XML[jconf_XML.find('>')+1: jconf_XML.rfind('<')].strip()
                authorsx = authors_XML[authors_XML.find('>')+1:
authors_XML.rfind('<')].strip()
                yearx = year_XML[year_XML.find('>')+1: year_XML.rfind('<')].strip()
                DOIx = DOI[DOI.find('>')+1: DOI.rfind('<')].strip()
                title_list.append(titlex)
                doi_list.append(DOIx)
```

```
disambiguated_author_profile.append([authorsx,titlex,yearx,orgx,jconfx,DOIx])
                xmlFile.write(pub_end_XML  + "\n")
            elif "<title>" in line and paper_index -1 in
predict_label_dict_fin[final_cluster_index]:
                title_XML = line
                xmlFile.write(title_XML  + "\n")
            elif "<abstract>" in line and paper_index -1 in
predict_label_dict_fin[final_cluster_index]:
                abstract_XML = line
                xmlFile.write(abstract_XML  +  "\n")
            elif "<organization>" in line and paper_index -1 in
predict_label_dict_fin[final_cluster_index]:
                organization_XML = line
                xmlFile.write(organization_XML  + "\n")
            elif "<jconf>" in line and paper_index -1 in
predict_label_dict_fin[final_cluster_index]:
                jconf_XML = line
                #print(jconf_XML)
                xmlFile.write(jconf_XML  + "\n")
            elif "<label>" in line and paper_index -1 in
predict_label_dict_fin[final_cluster_index]:
                label_XML = line
                xmlFile.write(label_XML  + "\n")
            elif "<pub_count>" in line and paper_index -1 in
predict_label_dict_fin[final_cluster_index]:
                pub_count_XML = line
                xmlFile.write(pub_count_XML  + "\n")
            elif "<authors>" in line and paper_index -1 in
predict_label_dict_fin[final_cluster_index]:
                authors_XML = line
                xmlFile.write(authors_XML  + "\n")
            elif "<year>" in line and paper_index -1 in
predict_label_dict_fin[final_cluster_index]:
                year_XML = line
                xmlFile.write(year_XML  + "\n")
            elif "<total_cnt>" in line:
                total_XML = line
                xmlFile.write(total_XML  + "\n")
            elif "<DOI>" in line:
                DOI = line
                xmlFile.write(DOI  + "\n")
            elif "<FullName>" in line:
                fullname_XML = line
                xmlFile.write(fullname_XML  + "\n")
                print("!!!!!EGO TAG IS",fullname_XML,"ego from Rdata XML file is", ego )
        #write the footer &
        #Checking if primary author has an ORCID paper set file to write into
Disambiguation XML as non-duplicate validated papers:
        title_set = set(title_list)
        doi_set = set(doi_list)
```

```
        ego = ego.replace(",",".")
        filepath_ORCID = "{}ORCID/{}_PRIMARY_ORCID.xml".format(file_path,ego)
        if os.path.isfile(filepath_ORCID):
            with open(filepath_ORCID, "r",encoding='utf-8') as filetoreadORCID:
                title_doi_instance = []
                for line in filetoreadORCID:
                    line = line.strip()
                    if "<publication>" in line:
                        #INCREMENT PAPER COUNTER - index
                        pub_start_XML = line
                        xmlFile.write(pub_start_XML + "\n")
                    elif "</publication>" in line:
                        pub_end_XML=line
                        titlex = title_XML[title_XML.find('>')+1:
title_XML.rfind('<')].strip()
                        orgx = organization_XML[organization_XML.find('>')+1:
organization_XML.rfind('<')].strip()
                        jconfx = jconf_XML[jconf_XML.find('>')+1:
jconf_XML.rfind('<')].strip()
                        authorsx = authors_XML[authors_XML.find('>')+1:
authors_XML.rfind('<')].strip()
                        yearx = year_XML[year_XML.find('>')+1: year_XML.rfind('<')].strip()
                        DOIx = DOI[DOI.find('>')+1: DOI.rfind('<')].strip()
                        if (not DOIx in doi_set) and (not titlex in title_set):
                            print("ORCID PAPER", DOIx, "is not in main papers will add
to",ego,"disambiguated paper set")
                            n_extraction += 1
                            paper_index += 1

disambiguated_author_profile.append([authorsx,titlex,yearx,orgx,jconfx,DOIx])
                            xmlFile.write(pub_end_XML  + "\n")
                        else:
                            print("ORCID PAPER", DOIx, "IS in main papers will NOT add
to",ego,"disambiguated paper set")
                    elif "<title>" in line:
                        title_XML = line
                        xmlFile.write(title_XML  + "\n")
                    elif "<abstract>" in line:
                        abstract_XML = line
                        xmlFile.write(abstract_XML  +  "\n")
                    elif "<organization>" in line:
                        organization_XML = line
                        xmlFile.write(organization_XML  + "\n")
                        #print(organization_XML)
                    elif "<jconf>" in line:
                        jconf_XML = line
                        #print(jconf_XML)
                        xmlFile.write(jconf_XML  + "\n")
                    elif "<label>" in line:
                        label_XML = line
                        xmlFile.write(label_XML  + "\n")
```

```
                    #print(label_XML)
                elif "<pub_count>" in line:
                    pub_count_XML = line
                    xmlFile.write(pub_count_XML  + "\n")
                    #print(pub_count_XML)
                elif "<authors>" in line:
                    authors_XML = line
                    xmlFile.write(authors_XML  + "\n")
                    #print(authors_XML)
                elif "<year>" in line:
                    year_XML = line
                    xmlFile.write(year_XML  + "\n")
                    #print(year_XML)
                elif "<total_cnt>" in line:
                    total_XML = line
                    xmlFile.write(total_XML  + "\n")
                elif "<DOI>" in line:
                    DOI = line
                    xmlFile.write(DOI  + "\n")
        #Addition of missing validated ORCID papers to researcher disambiguated papers
finished.
        xmlFile.write('</author_set>/n')


        print("finished conversion of disambiguated author cluster and addition of
valdiated missing ORCID papers to XML")
        print("Publications converted is",n_extraction, "compared with",
len(predict_label_dict_fin[final_cluster_index]), "papers in final cluster",
                "and a final cluster of:", predict_label_dict_fin[final_cluster_index])
        print("FINISHED XML CONVERSION OF DISAMBIGUATED AUTHOR", ego)
    disambiguated_author_profile = pd.DataFrame(disambiguated_author_profile, columns =
["Coauthors","Title", "Year","Organisation","Venue","DOI"])
    return n_extraction, disambiguated_author_profile
#Creating default XML Reading to Disambiguated XML function:
def default_XMLconvertion(file_path,x,ref_path,author_type,ego):
    #Now saving cluster index papers for primary author to XML file:
    xmlFileName = file_path + "disambiguated_main/" + str(x)
    filename_ref_default = ref_path  + str(x)
    filename_ref_default = filename_ref_default.split("_" + author_type + "_MAIN.xml")[0] +
'_' + author_type + '_REF.xml'

    with open(filename_ref_default, "r",encoding='utf-8') as filetoread, open(xmlFileName,
'w',encoding='utf-8') as xmlFile:
        # 这里是一行一行读取的文件
        print("Author ego is", ego, "extracting default reference author papers to XML:")
        xmlFile.write('<?xml version="1.0" encoding="UTF-8"?>\n')
        xmlFile.write('<author_set>\n')
        paper_index = 0
        n_extraction = 0
        for line in filetoread:
            line = line.strip()
```

```
#print("Stripped line",line)
if "<publication>" in line:
    #INCREMENT PAPER COUNTER - index
    pub_start_XML = line
    #print(pub_start_XML)
    paper_index += 1
    #print("paper_index incremented from", paper_index,"to",paper_index+1)
    xmlFile.write(pub_start_XML + "\n")
    n_extraction += 1
elif "</publication>" in line:
    pub_end_XML=line
    xmlFile.write(pub_end_XML  + "\n")
elif "<title>" in line:
    title_XML = line
    xmlFile.write(title_XML  + "\n")
elif "<abstract>" in line:
    abstract_XML = line
    xmlFile.write(abstract_XML  +  "\n")
#Note: need to add abstracts to ambiguious author XMLs?
elif "<organization>" in line:
    organization_XML = line
    xmlFile.write(organization_XML  + "\n")
    #print(organization_XML)
elif "<jconf>" in line:
    jconf_XML = line
    #print(jconf_XML)
    xmlFile.write(jconf_XML  + "\n")
elif "<label>" in line:
    label_XML = line
    xmlFile.write(label_XML  + "\n")
    #print(label_XML)
elif "<pub_count>" in line:
    pub_count_XML = line
    xmlFile.write(pub_count_XML  + "\n")
    #print(pub_count_XML)

elif "<authors>" in line:
    authors_XML = line
    xmlFile.write(authors_XML  + "\n")
    #print(authors_XML)
elif "<year>" in line:
    year_XML = line
    xmlFile.write(year_XML  + "\n")
    #print(year_XML)
elif "<total_cnt>" in line:
    total_XML = line
    xmlFile.write(total_XML  + "\n")
elif "<DOI>" in line:
    DOI = line
    xmlFile.write(DOI  + "\n")
elif "<FullName>" in line:
```

```
                fullname_XML = line
                xmlFile.write(fullname_XML  + "\n")
                print("!!!!!EGO TAG IS",fullname_XML,"ego from Rdata XML file is", ego )
                #write the footer
        xmlFile.write('</author_set>/n')
        print("finished conversion of default reference author cluster to XML")
        print("Publications converted is",n_extraction)
        print("FINISHED XML CONVERSION OF DISAMBIGUATED AUTHOR", ego)
    return n_extraction
```

Beginning of Xu et al (2018)'s adapted Author Disambiguation code:

```
class DataSet():
    """
pipeline for representation learning for all papers for a given name reference
"""

    def __init__(self, file_path, predicted_cluster_dict=None, cluster_index=None):
        if cluster_index != None:
            self.cluster_index=cluster_index
        else:
            self.cluster_index=False
        if predicted_cluster_dict != None:
            self.predicted_cluster_list = predicted_cluster_dict[cluster_index]
 #           print(self.predicted_cluster_list)
            for i in range(0,len(self.predicted_cluster_list)):
#                print(i)
                self.predicted_cluster_list[i] = self.predicted_cluster_list[i] + 1
  #          print("Cleaned",self.predicted_cluster_list)



        else:
            self.predicted_cluster_list = False
        self.file_path = file_path
        self.paper_authorlist_dict = {}
        self.paper_list = []
        self.coauthor_list = []
        self.paper_title=[]
        self.paper_jconf=[]
        self.paper_org = []
        self.paper_year = []
        self.paper_abstract = []
        self.paper_jconf_dict={}
        self.paper_title_dict = {}
        self.paper_org_dict = {}
        self.paper_year_dict = {}
        self.paper_abstract_dict = {}
        self.label_list = []
        self.C_Graph = nx.Graph()
```

```
        self.D_Graph = nx.Graph()
        self.T_Graph = nx.Graph()
        self.Jconf_Graph = nx.Graph()
        self.Org_Graph = nx.Graph()
        self.Year_Graph = nx.Graph()
        self.Abstract_Graph = nx.Graph()
        self.num_nnz = 0


    def reader_arnetminer(self, title_abstract_dict_original=False):
        #if title_abstract_dict_original == False:
        file_title_abstract =
open('PRIMARY_AUTHORS_XML/title_abstract_disseration_test.txt','r')
        #NOTE: A BUG IF encoding='utf-8'
        title_abstract_dict = {}
        for line in file_title_abstract.readlines():
            #Getting document of titles and abstracts divided by <>
            if "<>" in line:
                arr=line.split("<>")
                #splits by key and value - title/abstract - returning as [title - arr[0],
abstract arr[1]]
                #print(line)
                #print(arr)
                if len(arr)>1:
                    title_abstract_dict[arr[0].strip()]=arr[1].strip()
                    #print("arr length > 1", title_abstract_dict[arr[0]])
                    #print("Key", arr[0])
                else:
                    title_abstract_dict[arr[0].strip()]=""
                # if length of arr is not > 1 then only a title is present without
abstract.
        paper_index = 0
        coauthor_set = set()
        #Opening author XML file: Has title, author names, affilation etc in XML <>
        with open(self.file_path, "r",encoding='utf-8') as filetoread:
            for line in filetoread:
                line = line.strip()
                if "FullName" in line:
                    ego_name = line[line.find('>')+1:line.rfind('<')].strip()
             #If it is a title header then firstly obtain abstract using title as key
                #for abstract dict and secondly prep title with lemmetization and stemming.
                elif ("<title>" in line and  self.predicted_cluster_list==False) or
("<title>" in line and paper_index in self.predicted_cluster_list):
                    #Taking title from XML
                    text = line[line.find('>')+1: line.rfind('<')].strip()
                    abstract = ""
                    #Now moving to processing abstracts:
                    self.paper_abstract.append(paper_index)
                    if text in title_abstract_dict:
                        abstract = title_abstract_dict.get(text)
                        #then collect the abstract
```

```
                    if abstract=="":
                        abstract = text
                    abstract = re.sub("[:()'.,?`]", "",abstract)
                    abstract_list = []
                    #Abstract processed either as "" empty string or as abstract if
title_abstract_dict[text] not empty
                    if (len(abstract.split())>25):
                        #Taking the top five words from abstract with largest Term
Frequency-Inverse Document Frequency
                        # main idea of TF-IDF is: if a word appears frequently in a
document, that is, TF is high, and it rarely appears in other documents in the corpus,
                        #that is, the low of DF, that is, if the IDF is high, the term is
considered to have good classification capabilities.
                        abstract_list = analyse.extract_tags(abstract,topK=25)
                        #creating keyword tags for clustering
                    else:
                        #Otherwise, there is less than 25 words and we take them all.
                        abstract_list = abstract.split()
                    self.paper_abstract_dict[paper_index] = []
                    #Resetting for next iteration
                    for w in abstract_list:
                        try:
                            if w not in stopwords.words("english"):
                                #Making sure word w IS NOT a stopword as we do not want
these.
                                #Stop words are those words in natural language that have a
very little meaning, such as "is", "an", "the", etc.
                                #Search engines and other enterprise indexing platforms
often filter the stop words while fetching results from the database against
                                #the user queries. - e.g. An, the, has it, etc
                                #Stop words are often removed from the text before training
deep learning and machine learning models since stop words occur in abundance, hence
providing little to no unique information that can be used for classification or
clustering.
                                #initalizing
                                lemmatizer = WordNetLemmatizer()
                                #Lemmatization is the process of grouping together the
different inflected forms of a word so they can be analysed as a single item.
                                yx = lemmatizer.lemmatize(w.lower())
                                self.paper_abstract_dict[paper_index].append(yx)
                        except:
                            continue
                    #Lemmetized abstract of k =<=25 words
#                   Substiute
                    text = re.sub("[:()'.,?`]", "",text)
                    disease_List = text.split()
                #TITLES
                    #Now moving to processing titles
                    self.paper_title.append(paper_index)
                    self.paper_title_dict[paper_index]=[]
                    for w in disease_List:
```

```
                    try:
                        if w not in stopwords.words("english"):
                            #NOT a stopword as we do not want as,in,the etc
                            lemmatizer = WordNetLemmatizer()
                            yx = lemmatizer.lemmatize(w.lower())
                            self.paper_title_dict[paper_index].append(yx)
                    except:
                        continue
                elif ("<jconf>" in line and self.predicted_cluster_list==False) or
("<jconf>" in line and paper_index in self.predicted_cluster_list):
                    #print("Beginnning processing of conferences")
                    text = line[line.find('>') + 1: line.rfind('<')].strip()
                    text = re.sub("[:()'.,?`]", "", text)
                    disease_List = text.split()
                    self.paper_jconf.append(paper_index)
                    self.paper_jconf_dict[paper_index] = []
                    for w in disease_List:
                        if w not in stopwords.words("english"):
                            self.paper_jconf_dict[paper_index].append(w)
                        try:
                            if w not in stopwords.words("english"):
                                # 将每个单词转换为原型
                                lemmatizer = WordNetLemmatizer()
                                yx = lemmatizer.lemmatize(w.lower())
                                self.paper_jconf_dict[paper_index].append(yx)
                        except:
                            continue

# IF PUBLICATION XML HEADER THEN SIGNIFIES NEW PUBLICATION:
                elif "<publication>" in line:
                    #INCREMENT COUNTER
                    paper_index += 1
                    if self.predicted_cluster_list==False:
                        self.paper_list.append(paper_index) #List of papers indexs
                    elif self.predicted_cluster_list!=False and paper_index in
self.predicted_cluster_list:
                        self.paper_list.append(paper_index) #List of papers indexs

# BEGINNING YEAR PROCESSING - NOTE: NOT CURRENTLY USED IN PROOF-OF-CONCEPT.
                elif ("<year>" in line and self.predicted_cluster_list==False) or ("<year>"
in line and paper_index in self.predicted_cluster_list):
                    self.paper_year.append(paper_index)
                    self.paper_year_dict[paper_index] = []
                    text = line[line.find('>') + 1: line.rfind('<')].strip()
                    if text != "null":
                        self.paper_year_dict[paper_index].append(text)
# BEGINNING organization PROCESSING:
                elif ("<organization>" in line and self.predicted_cluster_list==False) or
("<organization>" in line and paper_index in self.predicted_cluster_list):
                    text = line[line.find('>') + 1: line.rfind('<')].strip()
                    text = re.sub("[:()'.,?`]", "", text)
```

```
                    disease_List = text.split()
                    self.paper_org.append(paper_index)
                    self.paper_org_dict[paper_index] = []
                    for w in disease_List:
                        try:
                            if w not in stopwords.words("english"):
                                # 将每个单词转换为原型
                                lemmatizer = WordNetLemmatizer()
                                yx = lemmatizer.lemmatize(w.lower())
                                self.paper_org_dict[paper_index].append(yx)
                        except:
                            continue
                elif ("<authors>" in line and self.predicted_cluster_list==False) or
("<authors>" in line and paper_index in self.predicted_cluster_list):
                    author_list = line[line.find('>')+1: line.rfind('<')].strip().split('
')
                    if len(author_list) > 1:
                        if ego_name in author_list:
                            author_list.remove(ego_name)
#                           #Show the list of collaborators in the first article
                            self.paper_authorlist_dict[paper_index] = author_list
                        else:
                            self.paper_authorlist_dict[paper_index] = author_list
                        for co_author in author_list:
                            coauthor_set.add(co_author)
                       # print("author_list",author_list)
#                         # Build a collaborator graph, only for the author collection of
each article
                        for pos in range(0, len(author_list) - 1):
                            for inpos in range(pos+1, len(author_list)):
                            #    print("pos", pos, "inpos", inpos)
#                                 # The result is a one-way partnership
                                src_node = author_list[pos]
                                dest_node = author_list[inpos]
                                #Returns True if the edge (u, v) is in the graph.
                                if not self.C_Graph.has_edge(src_node, dest_node):
                                    #If edge not yet made create edge with weight equal to
1
                                    self.C_Graph.add_edge(src_node, dest_node, weight = 1)
                                else:
                                    #Already an edge between authors - add 1 to weight of
collabs between two authors
                                    edge_weight =
self.C_Graph[src_node][dest_node]['weight']
#                                     # The more collaborations, the greater the weight of
joining
                                    edge_weight += 1
                                    self.C_Graph[src_node][dest_node]['weight'] =
edge_weight
                    else:
                        self.paper_authorlist_dict[paper_index] = []
```

```
                elif ("<label>" in line and self.predicted_cluster_list==False) or
("<label>" in line and paper_index in self.predicted_cluster_list):
                    label = int(line[line.find('>')+1: line.rfind('<')].strip())
                    self.label_list.append(label)
        self.coauthor_list = list(coauthor_set)        """
        compute the 2-extension coauthorship for each paper
        generate doc-doc network
        edge weight is based on 2-coauthorship relation
        edge weight details are in paper definition 3.3
        """
        paper_2hop_dict = {}
        for paper_idx in self.paper_list:
            temp = set()
            if self.paper_authorlist_dict[paper_idx] != []:
                for first_hop in self.paper_authorlist_dict[paper_idx]:
                    temp.add(first_hop)
                    if self.C_Graph.has_node(first_hop):
                        for snd_hop in self.C_Graph.neighbors(first_hop):
                            temp.add(snd_hop)
            paper_2hop_dict[paper_idx] = temp
        for i in self.paper_title:
            for j in self.paper_title:
                if i != j:
                    title_set1 = self.paper_title_dict[i]
                    title_set2 = self.paper_title_dict[j]
                    title_edge_weight =
len((set(title_set1)).intersection(set(title_set2)))
                    if title_edge_weight != 0:
                        self.T_Graph.add_edge(i,j,weight=title_edge_weight)
        for i in self.paper_jconf:
            for j in self.paper_jconf:
                if i != j:
                    jconf1 = self.paper_title_dict[i]
                    jconf2 = self.paper_title_dict[j]
                    jconf_edge_weight = len((set(jconf1)).intersection(set(jconf2)))
                    #print(title_edge_weight,"weight")
                    if jconf_edge_weight != 0:
                        self.Jconf_Graph.add_edge(i,j,weight=jconf_edge_weight)
        for i in self.paper_org:
            for j in self.paper_org:
                if i != j:
                    org1 = self.paper_org_dict[i]
                    org2 = self.paper_org_dict[j]
                    org_edge_weight = len((set(org1)).intersection(set(org2)))
                    if org_edge_weight != 0:
                        self.Org_Graph.add_edge(i,j,weight=org_edge_weight)
        for i in self.paper_year:
            for j in self.paper_year:
                if i != j:
                    year1 = self.paper_year_dict[i]
                    year2 = self.paper_year_dict[j]
```

```
                    year_edge_weight = 0
                    if not year1[0] == 'NA' and not year2[0] == 'NA':
                        if abs(int(year1[0])-int(year2[0]))<20:
                            year_edge_weight = 1
                        else:
                            year_edge_weight = 0
                    if year_edge_weight != 0:
                        self.Year_Graph.add_edge(i,j,weight=year_edge_weight)

        for i in self.paper_abstract:
            for j in self.paper_abstract:
                if i != j:
                    abstract1 = self.paper_abstract_dict[i]
                    abstract2 = self.paper_abstract_dict[j]
                    abstract_edge_weight =
len((set(abstract1)).intersection(set(abstract2)))
                    if abstract_edge_weight != 0:
                        self.Abstract_Graph.add_edge(i, j, weight=abstract_edge_weight)
        for idx1 in range(0, len(self.paper_list) - 1):
            for idx2 in range(idx1 + 1, len(self.paper_list)):
                temp_set1 = paper_2hop_dict[self.paper_list[idx1]]
                temp_set2 = paper_2hop_dict[self.paper_list[idx2]]
                edge_weight = len(temp_set1.intersection(temp_set2))
                #print(edge_weight, "weight")
                if edge_weight != 0:
                    self.D_Graph.add_edge(self.paper_list[idx1],
                                          self.paper_list[idx2],
                                          weight = edge_weight)
        bipartite_num_edge = 0
        for key, val in list(self.paper_authorlist_dict.items()):
            if val != []:
                bipartite_num_edge += len(val)
        self.num_nnz = self.D_Graph.number_of_edges() + \
                        self.T_Graph.number_of_edges() + \
                        self.Abstract_Graph.number_of_edges()+\
                        self.Org_Graph.number_of_edges()+\
                        self.C_Graph.number_of_edges()+\
                        self.Jconf_Graph.number_of_edges()+\
                            self.Year_Graph.number_of_edges() +\
                        bipartite_num_edge
        print(self.num_nnz,"+++++++")
        self.num_nnz = self.C_Graph.number_of_edges()
        self.num_nnz=300
        return ego_name
END>Dataset extraction and network construction
START> EMBEDDING OF NETWORKS
import math
def sigmoid(x):
     return float(1) / (1 + math.exp(-x))
import numpy as np
class BprOptimizer():
```

```
    """
    use Bayesian Personalized Ranking for objective loss
    Bayesian personalized ranking(BPR) , one of the famous learning to rank algorithms used
in recommender systems.
    latent_dimen: latent dimension
    alpha: learning rate
    matrix_reg: regularization parameter of matrix
    """
    def __init__(self, latent_dimen, alpha, matrix_reg):
        self.latent_dimen = latent_dimen
        self.alpha = alpha
        self.matrix_reg = matrix_reg

    def get_merge(self, dataset):
        node_dict = {}
        for node in dataset.Abstract_Graph.nodes():
            neighbors_list = dataset.Abstract_Graph.neighbors(node)
            if len(list(neighbors_list)) != 1:
                node_dict[node] = []
                for item in neighbors_list:
                    if len(dataset.Abstract_Graph.neighbors(item)) == 1:
                        node_dict[node].append(item)
        return node_dict

    def init_model(self, dataset):
        """
        initialize matrix using uniform [-0.2, 0.2]
        """
        self.paper_latent_matrix = {}
        self.author_latent_matrix = {}
        self.title_latent_matrix = {}
        self.jconf_latent_matrix = {}
        self.org_latent_matrix = {}
        self.year_latent_matrix = {}
        self.abstract_latent_matrix = {}
        # dd,dt,djconf,dorg,dyear,dabstract
        self.w=[1,1,0.1,0.1,0.1,1]
        node_dict = self.get_merge(dataset)
        in_list = []
        for key,val in node_dict.items():
            in_list=in_list+[key]
            in_list=in_list+val
            in_list=list(set(in_list))
        list_out = list(set(dataset.paper_list)-set(in_list))
        for item in list_out:
            self.paper_latent_matrix[item] = np.random.uniform(-0.5, 0.5,self.latent_dimen)
        for key ,val in node_dict.items():
            item_vec = np.random.uniform(-0.5, 0.5,self.latent_dimen)
            self.paper_latent_matrix[key] = item_vec
            for item in val:
                    self.paper_latent_matrix[item] = item_vec
```

```
    for paper_idx in dataset.paper_list:
        self.paper_latent_matrix[paper_idx] = np.random.uniform(-0.5, 0.5,
                                                                self.latent_dimen)
    for author_idx in dataset.coauthor_list:
        self.author_latent_matrix[author_idx] = np.random.uniform(-0.5, 0.5,
                                                                  self.latent_dimen)
    for title_idx in dataset.paper_title:
        self.title_latent_matrix[title_idx] = np.random.uniform(-0.5, 0.5,
                                                                  self.latent_dimen)
    for jconf_idx in dataset.paper_jconf:
        self.jconf_latent_matrix[jconf_idx] = np.random.uniform(-0.5, 0.5,
                                                                  self.latent_dimen)
    for org_idx in dataset.paper_org:
        self.org_latent_matrix[org_idx] = np.random.uniform(-0.5, 0.5,
                                                                  self.latent_dimen)
    for year_idx in dataset.paper_year:
        self.year_latent_matrix[year_idx] = np.random.uniform(-0.5, 0.5,
                                                                  self.latent_dimen)
    for abstract_idx in dataset.paper_abstract:
        self.abstract_latent_matrix[abstract_idx] = np.random.uniform(-0.5, 0.5,
                                                                  self.latent_dimen)

def update_pp_gradient(self, fst, snd, third):
    """
    SGD inference
    """
    x = self.predict_score(fst, snd, "pp") - \
        self.predict_score(fst, third, "pp")
    common_term = sigmoid(x) - 1

    grad_fst = common_term * (self.author_latent_matrix[snd] - \
                              self.author_latent_matrix[third]) + \
               2 * self.matrix_reg * self.author_latent_matrix[fst]
    self.author_latent_matrix[fst] = self.author_latent_matrix[fst] - \
                                     self.alpha * grad_fst

    grad_snd = common_term * self.author_latent_matrix[fst] + \
              2 * self.matrix_reg * self.author_latent_matrix[snd]
    self.author_latent_matrix[snd]= self.author_latent_matrix[snd] - \
                                    self.alpha * grad_snd

    grad_third = -common_term * self.author_latent_matrix[fst] + \
                 2 * self.matrix_reg * self.author_latent_matrix[third]
    self.author_latent_matrix[third] = self.author_latent_matrix[third] - \
                                       self.alpha * grad_third

def update_pd_gradient(self, fst, snd, third):
    x = self.predict_score(fst, snd, "pd") - \
        self.predict_score(fst, third, "pd")
    common_term = sigmoid(x) - 1
```

```
        grad_fst = common_term * (self.author_latent_matrix[snd] - \
                                  self.author_latent_matrix[third]) + \
                   2 * self.matrix_reg * self.paper_latent_matrix[fst]
        self.paper_latent_matrix[fst] = self.paper_latent_matrix[fst] - \
                                        self.alpha * grad_fst

        grad_snd = common_term * self.paper_latent_matrix[fst] + \
                   2 * self.matrix_reg * self.author_latent_matrix[snd]
        self.author_latent_matrix[snd]= self.author_latent_matrix[snd] - \
                                       self.alpha * grad_snd

        grad_third = -common_term * self.paper_latent_matrix[fst] + \
                     2 * self.matrix_reg * self.author_latent_matrix[third]
        self.author_latent_matrix[third] = self.author_latent_matrix[third] - \
                                           self.alpha * grad_third

    def update_dd_gradient(self, fst, snd, third):
        x = self.predict_score(fst, snd, "dd") - \
            self.predict_score(fst, third, "dd")
        common_term = sigmoid(x) - 1

        grad_fst = common_term * (self.paper_latent_matrix[snd] - \
                                  self.paper_latent_matrix[third]) + \
                   2 * self.matrix_reg * self.paper_latent_matrix[fst]
        self.paper_latent_matrix[fst] = self.paper_latent_matrix[fst] - \
                                        self.alpha * grad_fst * self.w[0]

        grad_snd = common_term * self.paper_latent_matrix[fst] + \
                   2 * self.matrix_reg * self.paper_latent_matrix[snd]
        self.paper_latent_matrix[snd]= self.paper_latent_matrix[snd] - \
                                       self.alpha * grad_snd* self.w[0]

        grad_third = -common_term * self.paper_latent_matrix[fst] + \
                     2 * self.matrix_reg * self.paper_latent_matrix[third]
        self.paper_latent_matrix[third] = self.paper_latent_matrix[third] - \
                                          self.alpha * grad_third* self.w[0]
    def update_dt_gradient(self, fst, snd, third):
        x = self.predict_score(fst, snd, "dt") - \
            self.predict_score(fst, third, "dt")
        common_term = sigmoid(x) - 1

        grad_fst = common_term * (self.title_latent_matrix[snd] - \
                                  self.title_latent_matrix[third]) + \
                   2 * self.matrix_reg * self.paper_latent_matrix[fst]
        self.paper_latent_matrix[fst] = self.paper_latent_matrix[fst] - \
                                        self.alpha * grad_fst* self.w[1]

        grad_snd = common_term * self.paper_latent_matrix[fst] + \
                   2 * self.matrix_reg * self.title_latent_matrix[snd]
        self.title_latent_matrix[snd]= self.title_latent_matrix[snd] - \
```

```
                                    self.alpha * grad_snd* self.w[1]

        grad_third = -common_term * self.paper_latent_matrix[fst] + \
                     2 * self.matrix_reg * self.title_latent_matrix[third]
        self.title_latent_matrix[third] = self.title_latent_matrix[third] - \
                                          self.alpha * grad_third* self.w[1]

    def update_djconf_gradient(self, fst, snd, third):
        x = self.predict_score(fst, snd, "djconf") - \
            self.predict_score(fst, third, "djconf")
        common_term = sigmoid(x) - 1

        grad_fst = common_term * (self.jconf_latent_matrix[snd] - \
                                  self.jconf_latent_matrix[third]) + \
                   2 * self.matrix_reg * self.paper_latent_matrix[fst]
        self.paper_latent_matrix[fst] = self.paper_latent_matrix[fst] - \
                                        self.alpha * grad_fst* self.w[2]

        grad_snd = common_term * self.paper_latent_matrix[fst] + \
                   2 * self.matrix_reg * self.jconf_latent_matrix[snd]
        self.jconf_latent_matrix[snd]= self.jconf_latent_matrix[snd] - \
                                       self.alpha * grad_snd* self.w[2]

        grad_third = -common_term * self.paper_latent_matrix[fst] + \
                     2 * self.matrix_reg * self.jconf_latent_matrix[third]
        self.jconf_latent_matrix[third] = self.jconf_latent_matrix[third] - \
                                          self.alpha * grad_third* self.w[2]

    def update_dorg_gradient(self, fst, snd, third):
        x = self.predict_score(fst, snd, "dorg") - \
            self.predict_score(fst, third, "dorg")
        common_term = sigmoid(x) - 1

        grad_fst = common_term * (self.org_latent_matrix[snd] - \
                                  self.org_latent_matrix[third]) + \
                   2 * self.matrix_reg * self.paper_latent_matrix[fst]
        self.paper_latent_matrix[fst] = self.paper_latent_matrix[fst] - \
                                        self.alpha * grad_fst* self.w[3]

        grad_snd = common_term * self.paper_latent_matrix[fst] + \
                   2 * self.matrix_reg * self.org_latent_matrix[snd]
        self.org_latent_matrix[snd] = self.org_latent_matrix[snd] - \
                                      self.alpha * grad_snd* self.w[3]

        grad_third = -common_term * self.paper_latent_matrix[fst] + \
                     2 * self.matrix_reg * self.org_latent_matrix[third]
        self.org_latent_matrix[third] = self.org_latent_matrix[third] - \
                                        self.alpha * grad_third* self.w[3]

    def update_dyear_gradient(self, fst, snd, third):
        x = self.predict_score(fst, snd, "dyear") - \
```

```
        self.predict_score(fst, third, "dyear")
    common_term = sigmoid(x) - 1

    grad_fst = common_term * (self.year_latent_matrix[snd] - \
                              self.year_latent_matrix[third]) + \
               2 * self.matrix_reg * self.paper_latent_matrix[fst]
    self.paper_latent_matrix[fst] = self.paper_latent_matrix[fst] - \
                                    self.alpha * grad_fst* self.w[4]

    grad_snd = common_term * self.paper_latent_matrix[fst] + \
               2 * self.matrix_reg * self.year_latent_matrix[snd]
    self.year_latent_matrix[snd] = self.year_latent_matrix[snd] - \
                                   self.alpha * grad_snd* self.w[4]

    grad_third = -common_term * self.paper_latent_matrix[fst] + \
                 2 * self.matrix_reg * self.year_latent_matrix[third]
    self.year_latent_matrix[third] = self.year_latent_matrix[third] - \
                                     self.alpha * grad_third* self.w[4]
def update_dabstract_gradient(self, fst, snd, third):
    x = self.predict_score(fst, snd, "dabstract") - \
        self.predict_score(fst, third, "dabstract")
    common_term = sigmoid(x) - 1

    grad_fst = common_term * (self.abstract_latent_matrix[snd] - \
                              self.abstract_latent_matrix[third]) + \
               2 * self.matrix_reg * self.paper_latent_matrix[fst]
    self.paper_latent_matrix[fst] = self.paper_latent_matrix[fst] - \
                                    self.alpha * grad_fst* self.w[5]

    grad_snd = common_term * self.paper_latent_matrix[fst] + \
               2 * self.matrix_reg * self.abstract_latent_matrix[snd]
    self.abstract_latent_matrix[snd] = self.abstract_latent_matrix[snd] - \
                                       self.alpha * grad_snd* self.w[5]

    grad_third = -common_term * self.paper_latent_matrix[fst] + \
                 2 * self.matrix_reg * self.abstract_latent_matrix[third]
    self.abstract_latent_matrix[third] = self.abstract_latent_matrix[third] - \
                                         self.alpha * grad_third* self.w[5]

def compute_pp_loss(self, fst, snd, third):
    """
    loss includes ranking loss and model complexity
    """
    x = self.predict_score(fst, snd, "pp") - \
         self.predict_score(fst, third, "pp")
    ranking_loss = -np.log(sigmoid(x))

    complexity = 0.0
    complexity += self.matrix_reg * np.dot(self.author_latent_matrix[fst],
                                           self.author_latent_matrix[fst])
    complexity += self.matrix_reg * np.dot(self.author_latent_matrix[snd],
```

```
                                        self.author_latent_matrix[snd])
    complexity += self.matrix_reg * np.dot(self.author_latent_matrix[third],
                                        self.author_latent_matrix[third])
    return ranking_loss + complexity

def compute_pd_loss(self, fst, snd, third):
    x = self.predict_score(fst, snd, "pd") - \
        self.predict_score(fst, third, "pd")
    ranking_loss = -np.log(sigmoid(x))

    complexity = 0.0
    complexity += self.matrix_reg * np.dot(self.paper_latent_matrix[fst],
                                        self.paper_latent_matrix[fst])
    complexity += self.matrix_reg * np.dot(self.author_latent_matrix[snd],
                                        self.author_latent_matrix[snd])
    complexity += self.matrix_reg * np.dot(self.author_latent_matrix[third],
                                        self.author_latent_matrix[third])
    return ranking_loss + complexity

def compute_dd_loss(self, fst, snd, third):
    x = self.predict_score(fst, snd, "dd") - \
        self.predict_score(fst, third, "dd")
    ranking_loss = -np.log(sigmoid(x))

    complexity = 0.0
    complexity += self.matrix_reg * np.dot(self.paper_latent_matrix[fst],
                                        self.paper_latent_matrix[fst])
    complexity += self.matrix_reg * np.dot(self.paper_latent_matrix[snd],
                                        self.paper_latent_matrix[snd])
    complexity += self.matrix_reg * np.dot(self.paper_latent_matrix[third],
                                        self.paper_latent_matrix[third])
    return ranking_loss + complexity

def compute_dt_loss(self, fst, snd, third):
    x = self.predict_score(fst, snd, "dt") - \
        self.predict_score(fst, third, "dt")
    ranking_loss = -np.log(sigmoid(x))

    complexity = 0.0
    complexity += self.matrix_reg * np.dot(self.paper_latent_matrix[fst],
                                        self.paper_latent_matrix[fst])
    complexity += self.matrix_reg * np.dot(self.title_latent_matrix[snd],
                                        self.title_latent_matrix[snd])
    complexity += self.matrix_reg * np.dot(self.title_latent_matrix[third],
                                        self.title_latent_matrix[third])
    return ranking_loss + complexity

def compute_djconf_loss(self, fst, snd, third):
    x = self.predict_score(fst, snd, "djconf") - \
        self.predict_score(fst, third, "djconf")
    ranking_loss = -np.log(sigmoid(x))
```

```
    complexity = 0.0
    complexity += self.matrix_reg * np.dot(self.paper_latent_matrix[fst],
                                           self.paper_latent_matrix[fst])
    complexity += self.matrix_reg * np.dot(self.jconf_latent_matrix[snd],
                                           self.jconf_latent_matrix[snd])
    complexity += self.matrix_reg * np.dot(self.jconf_latent_matrix[third],
                                           self.jconf_latent_matrix[third])
    return ranking_loss + complexity

def compute_dorg_loss(self, fst, snd, third):
    x = self.predict_score(fst, snd, "dt") - \
        self.predict_score(fst, third, "dt")
    ranking_loss = -np.log(sigmoid(x))

    complexity = 0.0
    complexity += self.matrix_reg * np.dot(self.paper_latent_matrix[fst],
                                           self.paper_latent_matrix[fst])
    complexity += self.matrix_reg * np.dot(self.org_latent_matrix[snd],
                                           self.org_latent_matrix[snd])
    complexity += self.matrix_reg * np.dot(self.org_latent_matrix[third],
                                           self.org_latent_matrix[third])
    return ranking_loss + complexity

def compute_dyear_loss(self, fst, snd, third):
    x = self.predict_score(fst, snd, "dyear") - \
        self.predict_score(fst, third, "dyear")
    ranking_loss = -np.log(sigmoid(x))

    complexity = 0.0
    complexity += self.matrix_reg * np.dot(self.paper_latent_matrix[fst],
                                           self.paper_latent_matrix[fst])
    complexity += self.matrix_reg * np.dot(self.year_latent_matrix[snd],
                                           self.year_latent_matrix[snd])
    complexity += self.matrix_reg * np.dot(self.year_latent_matrix[third],
                                           self.year_latent_matrix[third])
    return ranking_loss + complexity

def compute_dabstract_loss(self, fst, snd, third):
    x = self.predict_score(fst, snd, "dabstract") - \
        self.predict_score(fst, third, "dabstract")
    ranking_loss = -np.log(sigmoid(x))

    complexity = 0.0
    complexity += self.matrix_reg * np.dot(self.paper_latent_matrix[fst],
                                           self.paper_latent_matrix[fst])
    complexity += self.matrix_reg * np.dot(self.abstract_latent_matrix[snd],
                                           self.abstract_latent_matrix[snd])
    complexity += self.matrix_reg * np.dot(self.abstract_latent_matrix[third],
                                           self.abstract_latent_matrix[third])
    return ranking_loss + complexity
```

```
def predict_score(self, fst, snd, graph_type):
    """
    pp: person-person network
    pd: person-document bipartite network
    dd: doc-doc network
    detailed notation is inside paper
    """
    if graph_type == "pp":
        return np.dot(self.author_latent_matrix[fst],
                      self.author_latent_matrix[snd])
        # a = self.author_latent_matrix[fst]
        # b = self.author_latent_matrix[snd]
        # return np.dot(a,b)/float(np.sqrt(a.dot(a))*np.sqrt(b.dot(b)))
        # return np.sqrt(np.sum(np.square(self.author_latent_matrix[fst]
        #                                 - self.author_latent_matrix[snd])))
    elif graph_type == "pd":
        return np.dot(self.paper_latent_matrix[fst],
                      self.author_latent_matrix[snd])
        # a = self.paper_latent_matrix[fst]
        # b = self.author_latent_matrix[snd]
        # return np.dot(a, b) / float(np.sqrt(a.dot(a)) * np.sqrt(b.dot(b)))
        # return np.sqrt(np.sum(np.square(self.paper_latent_matrix[fst]
        #                                 - self.author_latent_matrix[snd])))
    elif graph_type == "dd":
        return np.dot(self.paper_latent_matrix[fst],
                      self.paper_latent_matrix[snd])
        # a = self.paper_latent_matrix[fst]
        # b = self.paper_latent_matrix[snd]
        # return np.dot(a, b) / float(np.sqrt(a.dot(a)) * np.sqrt(b.dot(b)))
        # return np.sqrt(np.sum(np.square(self.paper_latent_matrix[fst]
        #                                 - self.paper_latent_matrix[snd])))
    elif graph_type == "dt":
        return np.dot(self.paper_latent_matrix[fst],
                      self.title_latent_matrix[snd])
        # a = self.paper_latent_matrix[fst]
        # b = self.title_latent_matrix[snd]
        # return np.dot(a, b) / float(np.sqrt(a.dot(a)) * np.sqrt(b.dot(b)))
        # return np.sqrt(np.sum(np.square(self.paper_latent_matrix[fst]
        #                                 - self.title_latent_matrix[snd])))
    elif graph_type == "djconf":
        return np.dot(self.paper_latent_matrix[fst],
                      self.jconf_latent_matrix[snd])
        # a = self.paper_latent_matrix[fst]
        # b = self.jconf_latent_matrix[snd]
        # return np.dot(a, b) / float(np.sqrt(a.dot(a)) * np.sqrt(b.dot(b)))
        # return np.sqrt(np.sum(np.square(self.paper_latent_matrix[fst]
        #                                 - self.jconf_latent_matrix[snd])))
    elif graph_type == "dorg":
        return np.dot(self.paper_latent_matrix[fst],
                      self.org_latent_matrix[snd])
```

```
            # a = self.paper_latent_matrix[fst]
            # b = self.org_latent_matrix[snd]
            # return np.dot(a, b) / float(np.sqrt(a.dot(a)) * np.sqrt(b.dot(b)))
            # return np.sqrt(np.sum(np.square(self.paper_latent_matrix[fst]
            #                                 - self.org_latent_matrix[snd])))
        elif graph_type == "dyear":
            return np.dot(self.paper_latent_matrix[fst],
                          self.year_latent_matrix[snd])
            # a = self.paper_latent_matrix[fst]
            # b = self.year_latent_matrix[snd]
            # return np.dot(a, b) / float(np.sqrt(a.dot(a)) * np.sqrt(b.dot(b)))
            # return np.sqrt(np.sum(np.square(self.paper_latent_matrix[fst]
            #                                 - self.year_latent_matrix[snd])))
        elif graph_type == "dabstract":
            return np.dot(self.paper_latent_matrix[fst],
                          self.abstract_latent_matrix[snd])
            # a = self.paper_latent_matrix[fst]
            # b = self.abstract_latent_matrix[snd]
            # return np.dot(a, b) / float(np.sqrt(a.dot(a)) * np.sqrt(b.dot(b)))
            # return np.sqrt(np.sum(np.square(self.paper_latent_matrix[fst]
            #                                 - self.abstract_latent_matrix[snd])))
END> ---LOSS FUNCTION-PUBLICATION NETWORK EMBEDDING
START> ---SAMPLER-EMBEDDING
import random
def softmax(x):
    """Compute softmax values for each sets of scores in x."""
    if len(x)>0:
        e_x = np.exp(x - np.max(x))
        return e_x / e_x.sum(axis=0)
    else:
        return 0.5
"""
(i, j) belongs positive sample set
(i, t) belongs negative sample set
notation details are in the paper
"""

class CoauthorGraphSampler():
    @staticmethod
    def generate_triplet_uniform(dataset):
        """
        sample negative instance uniformly
        """
        a_i = random.choice(list(dataset.C_Graph.nodes()))
        a_t = random.choice(dataset.coauthor_list)
        while True:
            neig_list = list(dataset.C_Graph.neighbors(a_i))
            if a_t not in neig_list:
                # given a_i, sample its neighbor based on its weight value
                # idea of edge sampling
                weight_list = [dataset.C_Graph[a_i][nbr]['weight']
```

```
                        for nbr in neig_list]
            norm_weight_list = [float(w) / sum(weight_list)
                                for w in weight_list]
            a_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]
            yield a_i, a_j, a_t
            break

        else:
            a_i = random.choice(dataset.C_Graph.nodes())
            a_t = random.choice(dataset.coauthor_list)

@staticmethod
def generate_triplet_reject(dataset, bpr_optimizer):
    """
    generate negative instance using ranking-aware rejection sampler
    consider linear case
    """
    a_i = random.choice(dataset.C_Graph.nodes())
    neg_pair = random.sample(dataset.coauthor_list, 2)

    while True:
        neig_list = dataset.C_Graph.neighbors(a_i)
        if neg_pair[0] not in neig_list and neg_pair[1] not in neig_list:
            # given a_i, sample its neighbor based on its weight value
            # idea of edge sampling
            weight_list = [dataset.C_Graph[a_i][nbr]['weight']
                           for nbr in neig_list]
            norm_weight_list = [float(w) / sum(weight_list)
                                for w in weight_list]
            a_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

            # sample negative instance using ranking-aware rejection sampler
            sc1 = bpr_optimizer.predict_score(a_i, neg_pair[0], "pp")
            sc2 = bpr_optimizer.predict_score(a_i, neg_pair[1], "pp")
            a_t = neg_pair[0] if sc1 <= sc2 else neg_pair[1]
            yield a_i, a_j, a_t
            break

        else:
            a_i = random.choice(dataset.C_Graph.nodes())
            neg_pair = random.sample(dataset.coauthor_list, 2)

@staticmethod
def generate_triplet_adaptive(dataset, bpr_optimizer):
    """
    generate negative instance using adaptive sampling
    sample from a pre-defined exponential distribution
    """
    a_i = random.choice(dataset.C_Graph.nodes())
    neg_list = list(set(dataset.coauthor_list) - \
               set(dataset.C_Graph.neighbors(a_i)) - set([a_i]))
```

```
        # given a_i, sample its neighbor based on its weight value
        # idea of edge sampling
        neig_list = dataset.C_Graph.neighbors(a_i)
        weight_list = [dataset.C_Graph[a_i][nbr]['weight']
                       for nbr in neig_list]
        norm_weight_list = [float(w) / sum(weight_list)
                            for w in weight_list]
        a_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

        # sample negative instance based on pre-defined exponential distribution
        norm_soft = softmax([bpr_optimizer.predict_score(a_i, ne, "pp")
                             for ne in neg_list])
        a_t = np.random.choice(neg_list, 1, p = norm_soft)[0]
        yield a_i, a_j, a_t

class LinkedDocGraphSampler():
    @staticmethod
    def generate_triplet_uniform(dataset):
        d_i = random.choice(dataset.D_Graph.nodes())
        d_t = random.choice(dataset.paper_list)

        while True:
            neig_list = dataset.D_Graph.neighbors(d_i)
            if d_t not in neig_list:
                # given d_i, sample its neighbor based on its weight value
                # idea of edge sampling
                weight_list = [dataset.D_Graph[d_i][nbr]['weight']
                               for nbr in neig_list]
                norm_weight_list = [float(w) / sum(weight_list)
                                    for w in weight_list]
                d_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]
                yield d_i, d_j, d_t
                break

            else:
                d_i = random.choice(dataset.D_Graph.nodes())
                d_t = random.choice(dataset.paper_list)

    @staticmethod
    def generate_triplet_reject(dataset, bpr_optimizer):
        """
        generate negative instance using ranking-aware rejection sampler
        consider linear case
        """
        d_i = random.choice(dataset.D_Graph.nodes())
        neg_pair = random.sample(dataset.paper_list, 2)

        while True:
            neig_list = dataset.D_Graph.neighbors(d_i)
            if neg_pair[0] not in neig_list and neg_pair[1] not in neig_list:
```

```
                # given a_i, sample its neighbor based on its weight value
                # idea of edge sampling
                weight_list = [dataset.D_Graph[d_i][nbr]['weight']
                               for nbr in neig_list]
                norm_weight_list = [float(w) / sum(weight_list)
                                    for w in weight_list]
                d_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

                # sample negative instance using ranking-aware rejection sampler
                sc1 = bpr_optimizer.predict_score(d_i, neg_pair[0], "dd")
                sc2 = bpr_optimizer.predict_score(d_i, neg_pair[1], "dd")
                d_t = neg_pair[0] if sc1 <= sc2 else neg_pair[1]
                yield d_i, d_j, d_t
                break

            else:
                d_i = random.choice(dataset.D_Graph.nodes())
                neg_pair = random.sample(dataset.paper_list, 2)

    @staticmethod
    def generate_triplet_adaptive(dataset, bpr_optimizer):
        """
        generate negative instance using adaptive sampling
        sample from a pre-defined exponential distribution
        """
        d_i = random.choice(dataset.D_Graph.nodes())
        neg_list = list(set(dataset.paper_list) - \
                   set(dataset.D_Graph.neighbors(d_i)) - set([d_i]))

        # given a_i, sample its neighbor based on its weight value
        # idea of edge sampling
        neig_list = dataset.D_Graph.neighbors(d_i)
        weight_list = [dataset.D_Graph[d_i][nbr]['weight']
                       for nbr in neig_list]
        norm_weight_list = [float(w) / sum(weight_list)
                            for w in weight_list]
        d_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

        # sample negative instance based on pre-defined exponential distribution
        norm_soft = softmax([bpr_optimizer.predict_score(d_i, ne, "dd")
                             for ne in neg_list])
        d_t = np.random.choice(neg_list, 1, p = norm_soft)[0]
        yield d_i, d_j, d_t

class DocumentTitleSampler():
    @staticmethod
    def generate_triplet_uniform(dataset):
        t_i = random.choice(dataset.T_Graph.nodes())
        t_t = random.choice(dataset.paper_title)

        while True:
```

```
        neig_list = dataset.T_Graph.neighbors(t_i)
        if t_t not in neig_list:
            # given d_i, sample its neighbor based on its weight value
            # idea of edge sampling
            weight_list = [dataset.T_Graph[t_i][nbr]['weight']
                           for nbr in neig_list]
            norm_weight_list = [float(w) / sum(weight_list)
                                for w in weight_list]
            t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]
            yield t_i, t_j, t_t
            break
        else:
            t_i = random.choice(dataset.T_Graph.nodes())
            t_t = random.choice(dataset.paper_title)

@staticmethod
def generate_triplet_reject(dataset, bpr_optimizer):
    """
    generate negative instance using ranking-aware rejection sampler
    consider linear case
    """
    t_i = random.choice(dataset.T_Graph.nodes())
    neg_pair = random.sample(dataset.paper_title, 2)

    while True:
        neig_list = dataset.T_Graph.neighbors(t_i)
        if neg_pair[0] not in neig_list and neg_pair[1] not in neig_list:
            # given a_i, sample its neighbor based on its weight value
            # idea of edge sampling
            weight_list = [dataset.T_Graph[t_i][nbr]['weight']
                           for nbr in neig_list]
            norm_weight_list = [float(w) / sum(weight_list)
                                for w in weight_list]
            t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

            # sample negative instance using ranking-aware rejection sampler
            sc1 = bpr_optimizer.predict_score(t_i, neg_pair[0], "dt")
            sc2 = bpr_optimizer.predict_score(t_i, neg_pair[1], "dt")
            t_t = neg_pair[0] if sc1 <= sc2 else neg_pair[1]
            yield t_i, t_j, t_t
            break

        else:
            t_i = random.choice(dataset.T_Graph.nodes())
            neg_pair = random.sample(dataset.paper_title, 2)

@staticmethod
def generate_triplet_adaptive(dataset, bpr_optimizer):
    """
    generate negative instance using adaptive sampling
    sample from a pre-defined exponential distribution
```

```
        """
        t_i = random.choice(dataset.T_Graph.nodes())
        neg_list = list(set(dataset.paper_title) - \
                   set(dataset.T_Graph.neighbors(t_i)) - set([t_i]))

        # given a_i, sample its neighbor based on its weight value
        # idea of edge sampling
        neig_list = dataset.T_Graph.neighbors(t_i)
        weight_list = [dataset.T_Graph[t_i][nbr]['weight']
                       for nbr in neig_list]
        norm_weight_list = [float(w) / sum(weight_list)
                            for w in weight_list]
        t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

        # sample negative instance based on pre-defined exponential distribution
        norm_soft = softmax([bpr_optimizer.predict_score(t_i, ne, "dt")
                             for ne in neg_list])
        t_t = np.random.choice(neg_list, 1, p = norm_soft)[0]
        yield t_i, t_j, t_t

class DocumentJConfSampler():
    @staticmethod
    def generate_triplet_uniform(dataset):
        jconf_i = random.choice(dataset.Jconf_Graph.nodes())
        jconf_t = random.choice(dataset.paper_jconf)

        while True:
            neig_list = dataset.Jconf_Graph.neighbors(jconf_i)
            if jconf_t not in neig_list:
                weight_list = [dataset.Jconf_Graph[jconf_i][nbr]['weight']
                               for nbr in neig_list]
                norm_weight_list = [float(w) / sum(weight_list)
                                    for w in weight_list]
                jconf_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]
                yield jconf_i, jconf_j, jconf_t
                break

            else:
                jconf_i = random.choice(dataset.Jconf_Graph.nodes())
                jconf_t = random.choice(dataset.paper_jconf)

    @staticmethod
    def generate_triplet_reject(dataset, bpr_optimizer):
        """
        generate negative instance using ranking-aware rejection sampler
        consider linear case
        """
        jconf_i = random.choice(dataset.Jconf_Graph.nodes())
        neg_pair = random.sample(dataset.paper_jconf, 2)
        cnt=0
        while True:
```

```
            neig_list = dataset.Jconf_Graph.neighbors(jconf_i)
            if neg_pair[0] not in neig_list and neg_pair[1] not in neig_list:
                # given a_i, sample its neighbor based on its weight value
                # idea of edge sampling
                weight_list = [dataset.Jconf_Graph[jconf_i][nbr]['weight']
                               for nbr in neig_list]
                norm_weight_list = [float(w) / sum(weight_list)
                                    for w in weight_list]
                jconf_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

                # sample negative instance using ranking-aware rejection sampler
                sc1 = bpr_optimizer.predict_score(jconf_i, neg_pair[0], "djconf")
                sc2 = bpr_optimizer.predict_score(jconf_i, neg_pair[1], "djconf")
                jconf_t = neg_pair[0] if sc1 <= sc2 else neg_pair[1]
                yield jconf_i, jconf_j, jconf_t
                break

            else:
                cnt += 1
                if (cnt > 5):
                    neig_list = dataset.Jconf_Graph.neighbors(jconf_i)
                    weight_list = [dataset.Jconf_Graph[jconf_i][nbr]['weight']
                                   for nbr in neig_list]
                    norm_weight_list = [float(w) / sum(weight_list)
                                        for w in weight_list]
                    jconf_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]
                    jconf_t = random.choice(dataset.Jconf_Graph.nodes())
                    yield jconf_i, jconf_j, jconf_t
                    break

                jconf_i = random.choice(dataset.Jconf_Graph.nodes())
                neg_pair = random.sample(dataset.paper_jconf, 2)

    @staticmethod
    def generate_triplet_adaptive(dataset, bpr_optimizer):
        """
        generate negative instance using adaptive sampling
        sample from a pre-defined exponential distribution
        """
        jconf_i = random.choice(dataset.Jconf_Graph.nodes())
        neg_list = list(set(dataset.paper_jconf) - \
                   set(dataset.Jconf_Graph.neighbors(jconf_i)) - set([jconf_i]))

        # given a_i, sample its neighbor based on its weight value
        # idea of edge sampling
        neig_list = dataset.Jconf_Graph.neighbors(jconf_i)
        weight_list = [dataset.Jconf_Graph[jconf_i][nbr]['weight']
                       for nbr in neig_list]
        norm_weight_list = [float(w) / sum(weight_list)
                            for w in weight_list]
        jconf_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]
```

```
        # sample negative instance based on pre-defined exponential distribution
        jconf_t = 0
        if len(neg_list) > 0:
            norm_soft = softmax([bpr_optimizer.predict_score(jconf_i, ne, "djconf")
                                 for ne in neg_list])
            jconf_t = np.random.choice(neg_list, 1, p=norm_soft)[0]
        else:
            jconf_t = np.random.choice(neig_list)
        yield jconf_i, jconf_j, jconf_t


class BipartiteGraphSampler():
    @staticmethod
    def generate_triplet_uniform(dataset):
        d_i = random.choice(dataset.paper_list)
        a_t = random.choice(dataset.coauthor_list)

        while True:
            if dataset.paper_authorlist_dict[d_i] != [] \
                and a_t not in dataset.paper_authorlist_dict[d_i]:
                a_j = random.choice(dataset.paper_authorlist_dict[d_i])
                yield d_i, a_j, a_t
                break

            else:
                d_i = random.choice(dataset.paper_list)
                a_t = random.choice(dataset.coauthor_list)

    @staticmethod
    def generate_triplet_reject(dataset, bpr_optimizer):
        """
        generate negative instance using ranking-aware rejection sampler
        consider linear case
        """
        d_i = random.choice(dataset.paper_list)
        neg_pair = random.sample(dataset.coauthor_list, 2)

        while True:
            if dataset.paper_authorlist_dict[d_i] != [] \
                and neg_pair[0] not in dataset.paper_authorlist_dict[d_i] \
                    and neg_pair[1] not in dataset.paper_authorlist_dict[d_i]:

                a_j = random.choice(dataset.paper_authorlist_dict[d_i])

                # sample negative instance using ranking-aware rejection sampler
                sc1 = bpr_optimizer.predict_score(d_i, neg_pair[0], "pd")
                sc2 = bpr_optimizer.predict_score(d_i, neg_pair[1], "pd")
                a_t = neg_pair[0] if sc1 <= sc2 else neg_pair[1]
                yield d_i, a_j, a_t
                break
```

```
            else:
                d_i = random.choice(dataset.paper_list)
                neg_pair = random.sample(dataset.coauthor_list, 2)

    @staticmethod
    def generate_triplet_adaptive(dataset, bpr_optimizer):
        """
        generate negative instance using adaptive sampling
        sample from a pre-defined exponential distribution
        """
        d_i = random.choice(dataset.paper_list)
        neg_list = list(set(dataset.coauthor_list) - \
                        set(dataset.paper_authorlist_dict[d_i]))

        while True:
            if dataset.paper_authorlist_dict[d_i] != []:
                a_j = random.choice(dataset.paper_authorlist_dict[d_i])

                # sample negative instance based on pre-defined exponential distribution
                norm_soft = softmax([bpr_optimizer.predict_score(d_i, ne, "pd")
                                     for ne in neg_list])
                a_t = np.random.choice(neg_list, 1, p = norm_soft)[0]
                yield d_i, a_j, a_t
                break

            else:
                d_i = random.choice(dataset.paper_list)
                neg_list = list(set(dataset.coauthor_list) - \
                                set(dataset.paper_authorlist_dict[d_i]))

class DocumentOrgSampler():
    @staticmethod
    def generate_triplet_uniform(dataset):
        t_i = random.choice(dataset.Org_Graph.nodes())
        t_t = random.choice(dataset.paper_org)

        while True:
            neig_list = dataset.Org_Graph.neighbors(t_i)
            if t_t not in neig_list:
                # given d_i, sample its neighbor based on its weight value
                # idea of edge sampling
                weight_list = [dataset.Org_Graph[t_i][nbr]['weight']
                               for nbr in neig_list]
                norm_weight_list = [float(w) / sum(weight_list)
                                    for w in weight_list]
                t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]
                yield t_i, t_j, t_t
                break

            else:
```

```
            t_i = random.choice(dataset.Org_Graph.nodes())
            t_t = random.choice(dataset.paper_org)

    @staticmethod
    def generate_triplet_reject(dataset, bpr_optimizer):
        """
        generate negative instance using ranking-aware rejection sampler
        consider linear case
        """
        t_i = random.choice(dataset.Org_Graph.nodes())
        neg_pair = random.sample(dataset.paper_org, 2)
        cnt = 0
        while True:
            neig_list = dataset.Org_Graph.neighbors(t_i)
            if neg_pair[0] not in neig_list and neg_pair[1] not in neig_list:
                # given a_i, sample its neighbor based on its weight value
                # idea of edge sampling
                weight_list = [dataset.Org_Graph[t_i][nbr]['weight']
                               for nbr in neig_list]
                norm_weight_list = [float(w) / sum(weight_list)
                                    for w in weight_list]
                t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

                # sample negative instance using ranking-aware rejection sampler
                sc1 = bpr_optimizer.predict_score(t_i, neg_pair[0], "dorg")
                sc2 = bpr_optimizer.predict_score(t_i, neg_pair[1], "dorg")
                t_t = neg_pair[0] if sc1 <= sc2 else neg_pair[1]
                yield t_i, t_j, t_t
                break

            else:
                cnt += 1
                if (cnt > 5):
                    neig_list = dataset.Org_Graph.neighbors(t_i)
                    weight_list = [dataset.Org_Graph[t_i][nbr]['weight']
                                   for nbr in neig_list]
                    norm_weight_list = [float(w) / sum(weight_list)
                                        for w in weight_list]
                    t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]
                    t_t = random.choice(dataset.Org_Graph.nodes())
                    yield t_i, t_j, t_t
                    break
                t_i = random.choice(dataset.Org_Graph.nodes())
                neg_pair = random.sample(dataset.paper_org, 2)

    @staticmethod
    def generate_triplet_adaptive(dataset, bpr_optimizer):
        """
        generate negative instance using adaptive sampling
        sample from a pre-defined exponential distribution
        """
```

```
        t_i = random.choice(dataset.Org_Graph.nodes())
        neg_list = list(set(dataset.paper_org) - \
                   set(dataset.Org_Graph.neighbors(t_i)) - set([t_i]))

        # given a_i, sample its neighbor based on its weight value
        # idea of edge sampling
        neig_list = dataset.Org_Graph.neighbors(t_i)
        weight_list = [dataset.Org_Graph[t_i][nbr]['weight']
                       for nbr in neig_list]
        norm_weight_list = [float(w) / sum(weight_list)
                            for w in weight_list]
        t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

        # sample negative instance based on pre-defined exponential distribution
        t_t = 0
        if len(neg_list) > 0:
            norm_soft = softmax([bpr_optimizer.predict_score(t_i, ne, "dorg")
                                 for ne in neg_list])
            t_t = np.random.choice(neg_list, 1, p=norm_soft)[0]
        else:
            t_t = np.random.choice(neig_list)
        yield t_i, t_j, t_t

class DocumentYearSampler():
    @staticmethod
    def generate_triplet_uniform(dataset):
        t_i = random.choice(dataset.Year_Graph.nodes())
        t_t = random.choice(dataset.paper_year)

        while True:
            neig_list = dataset.Year_Graph.neighbors(t_i)
            if t_t not in neig_list:
                # given d_i, sample its neighbor based on its weight value
                # idea of edge sampling
                weight_list = [dataset.Year_Graph[t_i][nbr]['weight']
                               for nbr in neig_list]
                norm_weight_list = [float(w) / sum(weight_list)
                                    for w in weight_list]
                t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]
                yield t_i, t_j, t_t
                break

            else:
                t_i = random.choice(dataset.Year_Graph.nodes())
                t_t = random.choice(dataset.paper_year)

    @staticmethod
    def generate_triplet_reject(dataset, bpr_optimizer):
        """
        generate negative instance using ranking-aware rejection sampler
        consider linear case
```

```
    """
    t_i = random.choice(dataset.Year_Graph.nodes())
    neg_pair = random.sample(dataset.paper_year, 2)

    while True:
        neig_list = dataset.Year_Graph.neighbors(t_i)
        if neg_pair[0] not in neig_list and neg_pair[1] not in neig_list:
            # given a_i, sample its neighbor based on its weight value
            # idea of edge sampling
            weight_list = [dataset.Year_Graph[t_i][nbr]['weight']
                           for nbr in neig_list]
            norm_weight_list = [float(w) / sum(weight_list)
                                for w in weight_list]
            t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

            # sample negative instance using ranking-aware rejection sampler
            sc1 = bpr_optimizer.predict_score(t_i, neg_pair[0], "dyear")
            sc2 = bpr_optimizer.predict_score(t_i, neg_pair[1], "dyear")
            t_t = neg_pair[0] if sc1 <= sc2 else neg_pair[1]
            yield t_i, t_j, t_t
            break

        else:
            t_i = random.choice(dataset.Year_Graph.nodes())
            neg_pair = random.sample(dataset.paper_year, 2)

@staticmethod
def generate_triplet_adaptive(dataset, bpr_optimizer):
    """
    generate negative instance using adaptive sampling
    sample from a pre-defined exponential distribution
    """
    t_i = random.choice(dataset.Year_Graph.nodes())
    neg_list = list(set(dataset.paper_year) - \
               set(dataset.Year_Graph.neighbors(t_i)) - set([t_i]))

    # given a_i, sample its neighbor based on its weight value
    # idea of edge sampling
    neig_list = dataset.Year_Graph.neighbors(t_i)
    weight_list = [dataset.Year_Graph[t_i][nbr]['weight']
                   for nbr in neig_list]
    norm_weight_list = [float(w) / sum(weight_list)
                        for w in weight_list]
    t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

    # sample negative instance based on pre-defined exponential distribution
    norm_soft = softmax([bpr_optimizer.predict_score(t_i, ne, "dyear")
                         for ne in neg_list])
    t_t = np.random.choice(neg_list, 1, p = norm_soft)[0]
    yield t_i, t_j, t_t
```

```
class DocumentAbstractSampler():
    @staticmethod
    def generate_triplet_uniform(dataset):
        t_i = random.choice(dataset.Abstract_Graph.nodes())
        t_t = random.choice(dataset.paper_abstract)

        while True:
            neig_list = dataset.Abstract_Graph.neighbors(t_i)
            if t_t not in neig_list:
                # given d_i, sample its neighbor based on its weight value
                # idea of edge sampling
                weight_list = [dataset.Abstract_Graph[t_i][nbr]['weight']
                               for nbr in neig_list]
                norm_weight_list = [float(w) / sum(weight_list)
                                    for w in weight_list]
                t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]
                yield t_i, t_j, t_t
                break

            else:
                t_i = random.choice(dataset.Abstract_Graph.nodes())
                t_t = random.choice(dataset.paper_abstract)

    @staticmethod
    def generate_triplet_reject(dataset, bpr_optimizer):
        """
        generate negative instance using ranking-aware rejection sampler
        consider linear case
        """
        t_i = random.choice(dataset.Abstract_Graph.nodes())
        neg_pair = random.sample(dataset.paper_abstract, 2)
        cnt = 0
        while True:
            neig_list = dataset.Abstract_Graph.neighbors(t_i)
            if neg_pair[0] not in neig_list and neg_pair[1] not in neig_list:
                # given a_i, sample its neighbor based on its weight value
                # idea of edge sampling
                weight_list = [dataset.Abstract_Graph[t_i][nbr]['weight']
                               for nbr in neig_list]
                norm_weight_list = [float(w) / sum(weight_list)
                                    for w in weight_list]
                t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

                # sample negative instance using ranking-aware rejection sampler
                sc1 = bpr_optimizer.predict_score(t_i, neg_pair[0], "dabstract")
                sc2 = bpr_optimizer.predict_score(t_i, neg_pair[1], "dabstract")
                t_t = neg_pair[0] if sc1 <= sc2 else neg_pair[1]
                yield t_i, t_j, t_t
                break
            else:
                cnt+=1
```

```
                if(cnt>5):
                    neig_list = dataset.Abstract_Graph.neighbors(t_i)
                    weight_list = [dataset.Abstract_Graph[t_i][nbr]['weight']
                                   for nbr in neig_list]
                    norm_weight_list = [float(w) / sum(weight_list)
                                        for w in weight_list]
                    t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]
                    t_t = random.choice(dataset.Abstract_Graph.nodes())
                    yield t_i, t_j, t_t
                    break
                t_i = random.choice(dataset.Abstract_Graph.nodes())
                neg_pair = random.sample(dataset.paper_abstract, 2)

    @staticmethod
    def generate_triplet_adaptive(dataset, bpr_optimizer):
        """
        generate negative instance using adaptive sampling
        sample from a pre-defined exponential distribution
        """
        t_i = random.choice(dataset.Abstract_Graph.nodes())
        neg_list = list(set(dataset.paper_abstract) - \
                        set(dataset.Abstract_Graph.neighbors(t_i)) - set([t_i]))

        # given a_i, sample its neighbor based on its weight value
        # idea of edge sampling
        neig_list = dataset.Abstract_Graph.neighbors(t_i)
        weight_list = [dataset.Abstract_Graph[t_i][nbr]['weight']
                       for nbr in neig_list]
        norm_weight_list = [float(w) / sum(weight_list)
                            for w in weight_list]
        t_j = np.random.choice(neig_list, 1, p=norm_weight_list)[0]

        # sample negative instance based on pre-defined exponential distribution
        t_t = 0
        if len(neg_list) > 0:
            norm_soft = softmax([bpr_optimizer.predict_score(t_i, ne, "dabstract")
                                 for ne in neg_list])
            t_t = np.random.choice(neg_list, 1, p=norm_soft)[0]
        else:
            t_t = np.random.choice(neig_list)
        yield t_i, t_j, t_t
END> ---SAMPLER

def save_embedding(dict, paper_list, num_dimen,filename):
    """
    save the final embedding results for each document
    """
    fn = filename.split('/')[-1]
    embedding_file = open('C:\\Users\\Liam
Ephraims\\Author_Disambiguation\\emb'+fn+'.txt','w')
    # embedding_file.write(str(len(paper_list)) + ' ' + str(num_dimen) + os.linesep)
```

```
    D_matrix = dict[paper_list[0]]
    for idx in range(1, len(paper_list)):
        D_matrix = np.vstack((D_matrix, dict[paper_list[idx]]))
    D_matrix = np.hstack((np.array([list(range(1, len(paper_list)+1))]).T, D_matrix))
    np.savetxt(embedding_file, D_matrix[:,1:],
               fmt = ' '.join(['%1.5f'] * num_dimen))
TRAINER> ---TRAINER HELPER
class TrainHelper():
    @staticmethod
    def helper(num_epoch, dataset, bpr_optimizer,eval_f1, sampler_method,filename,
               pp_sampler=False,
               pd_sampler=False,
               dd_sampler=False,
               dt_sampler=False,
               djconf_sampler=False,
               dorg_sampler=False,
              dyear_sampler=False,
               dabstract_sampler=False, network_comparison=False,affinity_prop=False):

        bpr_optimizer.init_model(dataset)
        if sampler_method == "uniform":
            for _ in range(0, num_epoch):
                bpr_loss = 0.0
                for _ in range(0, dataset.num_nnz):
                    """
                    update embedding in person-person network
                    update embedding in person-document network
                    update embedding in doc-doc network
                    """
                    if not pp_sampler == False and dataset.C_Graph.number_of_edges() > 0:
                        for i, j, t in pp_sampler.generate_triplet_uniform(dataset):
                            bpr_optimizer.update_pp_gradient(i, j, t)
                            bpr_loss += bpr_optimizer.compute_pp_loss(i, j, t)

                    if not pd_sampler==False: #NOTE: NO GRAPH OBJECT USING PAPER LIST AND
CO AUTHOR LIST.
                        for i, j, t in pd_sampler.generate_triplet_uniform(dataset):
                            bpr_optimizer.update_pd_gradient(i, j, t)
                            bpr_loss += bpr_optimizer.compute_pd_loss(i, j, t)
                    if not dd_sampler == False and dataset.D_Graph.number_of_edges() > 0:
                        for i, j, t in dd_sampler.generate_triplet_uniform(dataset):
                            bpr_optimizer.update_dd_gradient(i, j, t)
                            bpr_loss += bpr_optimizer.compute_dd_loss(i, j, t)
                    if not dt_sampler == False:
                        for i, j, t in dt_sampler.generate_triplet_uniform(dataset):
                            bpr_optimizer.update_dt_gradient(i, j, t)
                            bpr_loss += bpr_optimizer.compute_dt_loss(i, j, t)
                    if not djconf_sampler==False and dataset.Jconf_Graph.number_of_edges()
> 0:
                       for i, j, t in djconf_sampler.generate_triplet_uniform(dataset):
                           bpr_optimizer.update_djconf_gradient(i, j, t)
```

```
                        bpr_loss += bpr_optimizer.compute_djconf_loss(i, j, t)
                    if not dorg_sampler == False and dataset.Org_Graph.number_of_edges() >
0:
                        for i, j, t in dorg_sampler.generate_triplet_uniform(dataset):
                            bpr_optimizer.update_dorg_gradient(i, j, t)
                            bpr_loss += bpr_optimizer.compute_dorg_loss(i, j, t)
                    if not dyear_sampler == False and dataset.Year_Graph.number_of_edges()
> 0:
                        for i, j, t in dyear_sampler.generate_triplet_uniform(dataset):
                            bpr_optimizer.update_dyear_gradient(i, j, t)
                            bpr_loss += bpr_optimizer.compute_dyear_loss(i, j, t)
                    if not dabstract_sampler == False and
dataset.Abstract_Graph.number_of_edges() > 0:
                        for i, j, t in dabstract_sampler.generate_triplet_uniform(dataset):
                            bpr_optimizer.update_dabstract_gradient(i, j, t)
                            bpr_loss += bpr_optimizer.compute_dabstract_loss(i, j, t)

                average_loss = float(bpr_loss) / dataset.num_nnz
                # print "average bpr loss is " + str(average_loss)
            average_f1,average_pre,average_rec, predict_label_dict,disambig_alg =
eval_f1.compute_f1(dataset, bpr_optimizer,network_comparison,affinity_prop)
            save_embedding(bpr_optimizer.paper_latent_matrix,
                           dataset.paper_list, bpr_optimizer.latent_dimen, filename)
            return average_f1,average_pre,average_rec, predict_label_dict,
dataset,disambig_alg,network_comparison,affinity_prop
        elif sampler_method == "reject":
            for _ in range(0, num_epoch):
                bpr_loss = 0.0
                for _ in range(0, dataset.num_nnz):
                    """
                    update embedding in person-person network
                    update embedding in person-document network
                    update embedding in doc-doc network
                    """
                    for i, j, t in pp_sampler.generate_triplet_reject(dataset,
bpr_optimizer):
                        bpr_optimizer.update_pp_gradient(i, j, t)
                        bpr_loss += bpr_optimizer.compute_pp_loss(i, j, t)
                    #
                    for i, j, t in pd_sampler.generate_triplet_reject(dataset,
bpr_optimizer):
                        bpr_optimizer.update_pd_gradient(i, j, t)
                        bpr_loss += bpr_optimizer.compute_pd_loss(i, j, t)

                    for i, j, t in dd_sampler.generate_triplet_reject(dataset,
bpr_optimizer):
                        bpr_optimizer.update_dd_gradient(i, j, t)
                        bpr_loss += bpr_optimizer.compute_dd_loss(i, j, t)

                    for i, j, t in
dt_sampler.generate_triplet_reject(dataset,bpr_optimizer):
```

```
                    bpr_optimizer.update_dt_gradient(i, j, t)
                    bpr_loss += bpr_optimizer.compute_dt_loss(i, j, t)

                for i, j, t in
djconf_sampler.generate_triplet_reject(dataset,bpr_optimizer):
                    bpr_optimizer.update_djconf_gradient(i, j, t)
                    bpr_loss += bpr_optimizer.compute_djconf_loss(i, j, t)
                #
                for i, j, t in
dorg_sampler.generate_triplet_reject(dataset,bpr_optimizer):
                    bpr_optimizer.update_dorg_gradient(i, j, t)
                    bpr_loss += bpr_optimizer.compute_dorg_loss(i, j, t)

                for i, j, t in
dyear_sampler.generate_triplet_reject(dataset,bpr_optimizer):
                    bpr_optimizer.update_dyear_gradient(i, j, t)
                    bpr_loss += bpr_optimizer.compute_dyear_loss(i, j, t)

                for i, j, t in
dabstract_sampler.generate_triplet_reject(dataset,bpr_optimizer):
                    bpr_optimizer.update_dabstract_gradient(i, j, t)
                    bpr_loss += bpr_optimizer.compute_dabstract_loss(i, j, t)

            average_f1, average_pre, average_rec = eval_f1.compute_f1(dataset,
bpr_optimizer,network_comparison,affinity_prop)
            # print 'f1 is ' + str(average_f1)
            return average_f1,average_pre,average_rec
        elif sampler_method == "adaptive":
            for _ in range(0, num_epoch):
                bpr_loss = 0.0
                for _ in range(0, dataset.num_nnz):
                    """
                    update embedding in person-person network
                    update embedding in person-document network
                    update embedding in doc-doc network
                    """
                    for i, j, t in pp_sampler.generate_triplet_adaptive(dataset,
bpr_optimizer):
                        bpr_optimizer.update_pp_gradient(i, j, t)
                        bpr_loss += bpr_optimizer.compute_pp_loss(i, j, t)
                    #
                    for i, j, t in pd_sampler.generate_triplet_adaptive(dataset,
bpr_optimizer):
                        bpr_optimizer.update_pd_gradient(i, j, t)
                        bpr_loss += bpr_optimizer.compute_pd_loss(i, j, t)

                    for i, j, t in dd_sampler.generate_triplet_adaptive(dataset,
bpr_optimizer):
                        bpr_optimizer.update_dd_gradient(i, j, t)
                        bpr_loss += bpr_optimizer.compute_dd_loss(i, j, t)
```

```
                    for i, j, t in
dt_sampler.generate_triplet_adaptive(dataset,bpr_optimizer):
                        bpr_optimizer.update_dt_gradient(i, j, t)
                        bpr_loss += bpr_optimizer.compute_dt_loss(i, j, t)

                    for i, j, t in
djconf_sampler.generate_triplet_adaptive(dataset,bpr_optimizer):
                        bpr_optimizer.update_djconf_gradient(i, j, t)
                        bpr_loss += bpr_optimizer.compute_djconf_loss(i, j, t)

                    for i, j, t in
dorg_sampler.generate_triplet_adaptive(dataset,bpr_optimizer):
                        bpr_optimizer.update_dorg_gradient(i, j, t)
                        bpr_loss += bpr_optimizer.compute_dorg_loss(i, j, t)

                    for i, j, t in
dyear_sampler.generate_triplet_adaptive(dataset,bpr_optimizer):
                       bpr_optimizer.update_dyear_gradient(i, j, t)
                       bpr_loss += bpr_optimizer.compute_dyear_loss(i, j, t)

                    for i, j, t in
dabstract_sampler.generate_triplet_adaptive(dataset,bpr_optimizer):
                        bpr_optimizer.update_dabstract_gradient(i, j, t)
                        bpr_loss += bpr_optimizer.compute_dabstract_loss(i, j, t)

            average_f1, average_pre, average_rec = eval_f1.compute_f1(dataset,
bpr_optimizer,network_comparison,affinity_prop)
            # print 'f1 is ' + str(average_f1)
            return average_f1,average_pre,average_rec
        save_embedding(bpr_optimizer.paper_latent_matrix,
                       dataset.paper_list, bpr_optimizer.latent_dimen,filename)
TRAINER END> ---TRAINER HELPER---

START> CLUSTERING - XMEANS
from sklearn.cluster import *
from sklearn.decomposition import *
from sklearn.externals import joblib
def construct_doc_matrix(dict, paper_list):
    """
    construct the learned embedding for document clustering
    dict: {paper_index, numpy_array}
    """
    D_matrix = dict[paper_list[0]]
    for idx in range(1, len(paper_list)):
        D_matrix = np.vstack((D_matrix, dict[paper_list[idx]]))
    return D_matrix

from scipy import stats
from sklearn.cluster import KMeans

class XMeans(KMeans):
```

```
    def __init__(self, k_init = 1, **k_means_args):
        self.k_init = k_init
        self.k_means_args = k_means_args

    def fit(self, X):
        km = KMeans(self.k_init, **self.k_means_args).fit(X)
        self.cluster_centers_ = list(km.cluster_centers_)
        self.labels_, _ = self.__recrucive_clustering(X, km.labels_, np.unique(km.labels_),
np.max(km.labels_))
        self.cluster_centers_ = np.array(self.cluster_centers_)
        return self

    def __recrucive_clustering(self, X, labels, labelset, splits):
        for label in labelset:
            cluster = X[labels == label]
            if len(cluster) <= 3: continue
            km = KMeans(2, **self.k_means_args).fit(cluster)
            if self.__recrucive_decision(cluster, km.labels_, km.cluster_centers_):
                self.cluster_centers_[label] = km.cluster_centers_[0]
                self.cluster_centers_.append(km.cluster_centers_[1])
                km.labels_[km.labels_ == 1] += splits
                km.labels_[km.labels_ == 0] += label
                labels[labels == label], splits = self.__recrucive_clustering(cluster,
km.labels_, [label, splits+1], splits+1)
        return labels, splits

    def __recrucive_decision(self, cluster, labels, centers):
        samples = cluster.shape[0]
        parameters = cluster.shape[1] * (cluster.shape[1] + 3) / 2
        bic = self.__bic(cluster, None, samples, parameters)
        new_bic = self.__bic(cluster[labels == 0], cluster[labels == 1], samples,
parameters*2)
        return bic > new_bic

    def __bic(self, cluster_0, cluster_1, samples, parameters):
        if cluster_1 is not None:
            alpha = self.__model_likelihood(cluster_0, cluster_1)
            return -2.0 * (samples * np.log(alpha) + self.__log_likelihood(cluster_0) +
self.__log_likelihood(cluster_1)) + parameters * np.log(samples)
        else:
            return -2.0 * self.__log_likelihood(cluster_0) + parameters * np.log(samples)

    def __model_likelihood(self, cluster_0, cluster_1):
        mu_0, mu_1 = np.mean(cluster_0, axis=0), np.mean(cluster_1, axis=0)
        sigma_0, sigma_1 = np.cov(cluster_0.T) , np.cov(cluster_1.T)
        det_0 = np.linalg.det(sigma_0) if len(cluster_0) > 1 else 0.0
        det_1 = np.linalg.det(sigma_1)if len(cluster_1) > 1 else 0.0
        beta = np.linalg.norm(mu_0 - mu_1) / np.sqrt(det_0 + det_1+0.00001)
        return 0.5 / stats.norm.cdf(beta)
```

```
    def __log_likelihood(self, cluster):
        if len(cluster) == 1: return np.log(1.0)
        mu = np.mean(cluster, axis=0)
        sigma = np.cov(cluster.T)
        if not str(sigma[0][0]).isdigit():
            return np.log(1.0)
        try:
            log_likehood = np.sum(stats.multivariate_normal.logpdf(x, mu, sigma) for x in
cluster)
        except:
            sigma = sigma * np.identity(sigma.shape[0])
            log_likehood = np.sum(stats.multivariate_normal.logpdf(x, mu, sigma) for x in
cluster)
        return log_likehood

    def predict(self):
        return self.labels_

    def fit_predict(self, X):
        self.fit(X)
        return self.labels_
```

```
START> ---MODEL READER
```

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

from joblib import load

class Token:
    __slots__ = ('text', 'span')

    def __init__(self, text, span):
        self.text = text
        self.span = span

    def __repr__(self):
        return '<Token %s>' % self.text.encode('utf-8')

    def __reduce__(self):
        return (self.__class__, (self.text, self.span))


class RegexpTokenizer(object):
    __slots__ = ('_rule',)

    def __init__(self, rule=r'[\w\d]+'):
        self._rule = re.compile(rule, re.UNICODE)

    def tokenize(self, text):
```

```
        return [Token(text[o.start():o.end()], o.span()) for o in
self._rule.finditer(text)]

class ModelReader(object):
    def __init__(self, model_file):
        model = load(model_file, mmap_mode='r')
        self._word_embedding = model['word_embedding']
        self._entity_embedding = model['entity_embedding']
        self._W = model.get('W')
        self._b = model.get('b')
        self._vocab = model.get('vocab')

        self._tokenizer = RegexpTokenizer()

    @property
    def vocab(self):
        return self._vocab

    @property
    def word_embedding(self):
        return self._word_embedding

    @property
    def entity_embedding(self):
        return self._entity_embedding

    @property
    def W(self):
        return self._W

    @property
    def b(self):
        return self._b

    def get_word_vector(self, word, default=None):
        index = self._vocab.get_word_index(word)
        if index:
            return self.word_embedding[index]
        else:
            return self.word_embedding[0]
    def get_entity_vector(self, title, default=None):
        index = self._vocab.get_entity_index(title)
        if index:
            return self.entity_embedding[index]
        else:
            return default

    def get_text_vector(self, text):
        vectors = [self.get_word_vector(t.text.lower())
                   for t in self._tokenizer.tokenize(text)]
        vectors = [v for v in vectors if v is not None]
```

```
        if not vectors:
            return None

        ret = np.mean(vectors, axis=0)
        ret = np.dot(ret, self._W)
        ret += self._b

        ret /= np.linalg.norm(ret, 2)

        return ret
END> ---MODEL READER
START>- EVAL_METRIC—AND CLUSTERING
def hdbscan(D_matrix):
    return HDBSCAN(min_cluster_size=1).fit_predict(D_matrix)
def dbscan(D_matrix):
    return DBSCAN(eps=1.5, min_samples=2).fit_predict(D_matrix)
def ap(D_matrix):
    return AffinityPropagation(damping=0.6).fit_predict(D_matrix)
def xmeans(D_matrix):
    return XMeans(random_state=1).fit_predict(D_matrix)
def meanshift(D_matrix):
    # bandwidth=estimate_bandwidth(D_matrix,quantile=0.4)
    return MeanShift().fit_predict(D_matrix)
def kmeans(D_matrix,true_cluster_size):
    return KMeans(n_clusters=true_cluster_size,init="k-means++").fit_predict(D_matrix)
def hac(D_matrix,true_cluster_size):
    return AgglomerativeClustering(n_clusters = true_cluster_size,linkage =
"average",affinity = "cosine").fit_predict(D_matrix)
def compute_distance(D_matrix,predict_label_dict):
    avg_distance = 0
    sum_list = []
    for key, value in predict_label_dict.items():
        list_item = []
        for j in value:
            list_item.append(D_matrix[j])
        D_itme_array = np.array(list_item)
        for i in range(D_itme_array.shape[0]):
            sum_distance = 0
            for j in range(D_itme_array.shape[0]):
                dist = np.sqrt(np.sum(np.square(D_itme_array[i] - D_itme_array[j])))
                sum_distance += dist
            sum_list.append(sum_distance)
        avg_distance = float(np.median(sum_list)) / D_itme_array.shape[0]
        break
    return avg_distance
def compute_sd(D_matrix,predict_label_dict):
    D_matrix = D_matrix
    dstd = np.std(D_matrix)
    dvar = D_matrix.shape[0]*dstd*dstd
    sum_ci_var = 0.0
    center = []
```

```
    for key ,val in predict_label_dict.items():
        C_i_matrix = D_matrix[0]
        for i in val:
            C_i_matrix=np.row_stack((C_i_matrix,D_matrix[i-1]))
        np.delete(C_i_matrix,0,axis=0)
        c_i_std = np.std(C_i_matrix)
        c_i_var = C_i_matrix.shape[0]*c_i_std*c_i_std
        sum_ci_var+=c_i_std
        center.append(np.sum(C_i_matrix,axis=0)/float(C_i_matrix.shape[0]))
    scat = sum_ci_var/float(len(predict_label_dict)*dvar)
    return scat

class Evaluator():
    @staticmethod
    def compute_f1(dataset, bpr_optimizer, network_comparison, affinity_prop):
        D_matrix =
construct_doc_matrix(bpr_optimizer.paper_latent_matrix,dataset.paper_list)
       #print(D_matrix)
        X_embedded = TSNE(n_components=2).fit_transform(D_matrix)
        x = []
        y = []
        for i in range(X_embedded.shape[0]):
            x.append(X_embedded[i][0])
            y.append(X_embedded[i][1])
        plt.scatter(x, y)
        plt.show()
        true_cluster_size = len(set(dataset.label_list))
        true_label_dict = {}
        for idx, true_lbl in enumerate(dataset.label_list):
            if true_lbl not in true_label_dict:
                true_label_dict[true_lbl] = [idx]
            else:
                true_label_dict[true_lbl].append(idx)
        predict_label_dict = {}
        y_pred = dbscan(D_matrix)
        print("Y_pred", y_pred)
        for idx, pred_lbl in enumerate(y_pred):
            if pred_lbl not in predict_label_dict:
                predict_label_dict[pred_lbl] = [idx]
            else:
                predict_label_dict[pred_lbl].append(idx)

        y_pred1 = ap(D_matrix)
        predict_label_dict1 = {}
        for idx, pred_lbl in enumerate(y_pred1):
            if pred_lbl not in predict_label_dict1:
                predict_label_dict1[pred_lbl] = [idx]
            else:
                predict_label_dict1[pred_lbl].append(idx)
        avg_distance1 = compute_distance(D_matrix,predict_label_dict1)
        sd1 = compute_sd(D_matrix,predict_label_dict1)
```

```
        y_pred2 = xmeans(D_matrix)
        predict_label_dict2={}
        for idx, pred_lbl in enumerate(y_pred2):
            if pred_lbl not in predict_label_dict2:
                predict_label_dict2[pred_lbl] = [idx]
            else:
                predict_label_dict2[pred_lbl].append(idx)
        sd2 = compute_sd(D_matrix, predict_label_dict2)
        #NOW ONLY USING THE DBSCAN CLUSTERING ALGORITHM OUTPUT IN ADAPTED APPROACH!!!!!!!!
        if np.all(y_pred==y_pred2):
        #then x-means and DBSCAN are both in agreeance
             predict_label_dict = predict_label_dict
             print("...Using DBSCAN clustering result...XMEANS=DBSCAN!")
             disambig_alg = "DBSCAN"
        elif sd2>sd1 and avg_distance1>1.75 and network_comparison == True and
affinity_prop == True:
             predict_label_dict=predict_label_dict2
             print("...Using affinity prop clustering result...")
             disambig_alg = "Affinity Propagation"
        else:
             predict_label_dict = predict_label_dict
             print("...Using DBSCAN clustering result...Default!")
             disambig_alg = "DBSCAN"

   #NOTE: NOW ONLY USING DBSCAN as clustering algorithm instead of affinity propagation and
xmeans.
        # compute cluster-level F1
        # let's denote C(r) as clustering result and T(k) as partition (ground-truth)
        # construct r * k contingency table for clustering purpose
        r_k_table = []
        for v1 in predict_label_dict.values():
            k_list = []

            for v2 in true_label_dict.values():

                N_ij = len(set(v1).intersection(v2))
                k_list.append(N_ij)
            r_k_table.append(k_list)
        r_k_matrix = np.array(r_k_table)
        r_num = int(r_k_matrix.shape[0])
        # compute F1 for each row C_i
        #print(r_k_table)

        sum_f1 = 0.0
        sum_pre = 0.0
        sum_rec = 0.0
        for row in range(0, r_num):
            row_sum = np.sum(r_k_matrix[row,:])
            if row_sum != 0:
                max_col_index = np.argmax(r_k_matrix[row,:])
                row_max_value = r_k_matrix[row, max_col_index]
```

```
                prec = float(row_max_value) / row_sum
                col_sum = np.sum(r_k_matrix[:, max_col_index])
                rec = float(row_max_value) / col_sum
                row_f1 = float(2 * prec * rec) / (prec + rec)
                sum_f1 += row_f1
                sum_pre += prec
                sum_rec += rec
        # print len(y_pred)
        average_f1 =float(sum_f1) / r_num
        #average_f1 = float(sum_f1)
        average_pre = float(sum_pre) /r_num
        average_rec = float(sum_rec) /r_num
        return average_f1,average_pre,average_rec, predict_label_dict,disambig_alg

END> ---EVAL_METRIC
NOTE: THE ABOVE SECTION ALLOWS FOR AUTOMATIC HYPERPARAMETER TUNING ON RESEARCH ORGANISATION
RESEARCHER DATASETS WHERE DOIs ARE STRUCTURED WITHIN A RELATIONAL DATASET.

START> ---MAIN---
#nltk.download('wordnet')
#if __name__ == "__main__":
#Getting list of author xml files from sample_data folder, using os.walk, returns a nested
list, which
#is then collected using file_list[0] to obtain list unnested.
def get_median(data):
        data = sorted(data)
        size = len(data)
        if size % 2 == 0:
            median = (data[size//2]+data[size//2-1])/2
            data[0] = median
        if size % 2 == 1:
            median = data[(size-1)//2]
            data[0] = median
        return data[0]
F1_List = []
F1_Max_List = []
F1_Max_List_pre =[]
F1_Max_List_rec = []
f1_name = {}
def get_file_list(file_dir):
    file_list = []
    for root, dirs, files in os.walk(file_dir):
        file_list.append(files)
    return file_list[0]
import time
def primary_or_secondary_main(file_path, ref_path,author_type,network_comparison=False,
affinity_prop=False):
    place_holder = 0
    author_summary_dataframe = []
    failed_authors = 0
    disambiguated_authors = 0
```

```
    defaulted_reference_authors = 0
    main_count_dict = check_main_count()
    ref_count_dict = check_ref_count()
    file_path_list = file_path + "MAIN/"
    file_list = get_file_list(file_path_list)
    #Sorting file list alphabetically
    file_list = sorted(file_list)
    file_list = file_list[:]
    #counter
    cnt = 0
    copy_f1_list = []
    #Begins iteration for each author file going into main function.
    #Extracting primary authors:
    for x in file_list:
        final_count = 0
        cnt += 1
        filename = file_path + "MAIN/" + str(x)
        F1_Max_List = []
        #Disambiugation rule set and status reports:
        #Create a author_total_cnt_dict which records authors n papers for ref and main.
        #Disambiguation Rule 1: main(n) > 1 & ref(n) > 1 (Can attempt disambiguation)
Status: Disambiguated (k clusters)|Disambiguated (1 cluster)
        #Disambiguation Rule 2: main(n) > 1 & ref(n) <= 1 & cluster(n) == 1 (Will take
whole cluster - network comparison to ref not undertaken) Status: Disambiguated (1 cluster)
        #Else: Take reference network as authors paper set - provided by uni and verified
ORCID profiles Status: Default Reference Network
        #If no main papers for author return: None Status: No author main network file
        #If no reference papers file for author: Status: No author reference file
        latent_dimen = 40
        alpha = 0.1
        matrix_reg = 0.1
        num_epoch = 10
        sampler_method = 'uniform'
        dataset = DataSet(filename)
        ego = dataset.reader_arnetminer()
        if ego in main_count_dict:
            main_counts = main_count_dict[ego]
        else:
            main_counts = 0
        if ego in ref_count_dict:
            ref_counts = ref_count_dict[ego]
        else:
            ref_counts = 0

        print("Main paper number for author", ego, "is",main_counts)
        print("Ref paper number for author", ego, "is",ref_counts)
        if main_counts > 1:
            bpr_optimizer = BprOptimizer(latent_dimen, alpha, matrix_reg)
            dd_sampler = LinkedDocGraphSampler()
            dt_sampler = DocumentTitleSampler()
            djconf_sampler = DocumentJConfSampler()
```

```
            pp_sampler = CoauthorGraphSampler()
            pd_sampler = BipartiteGraphSampler()
            dyear_sampler = DocumentYearSampler()
            dorg_sampler = DocumentOrgSampler()
            dabstract_sampler = DocumentAbstractSampler()
            eval_f1 = Evaluator()
            run_helper = TrainHelper()
            avg_f1,avg_pre,avg_rec,predict_label_dict_fin,
graph,disambig_alg,network_comparison,affinity_prop = run_helper.helper(num_epoch, dataset,
bpr_optimizer,
                eval_f1, sampler_method,filename, pp_sampler,pd_sampler,dd_sampler,
                dt_sampler,
               djconf_sampler,dorg_sampler,
              dyear_sampler,
               dabstract_sampler,network_comparison=False,affinity_prop=False)
            inexact_graph_edit_dist_cluster_dict = {}
            print(predict_label_dict_fin)
            cluster_count = len(predict_label_dict_fin)
            print(cluster_count,"cluster count")
        else:
            print("Disambiguation aborted: Main paper error - count is",main_counts)
            cluster_count = 0

            #Extracting primary authors reference networks:
            print("Attempting network comparison; reference network setup")
        if author_type == "PRIMARY" and ((main_counts>1 and ref_counts>1) or
(cluster_count==1 and main_counts>1 and ref_counts<=1)) :
            print("Passed network comparison requirements:", "main
count:",main_counts,"ref_count:",ref_counts,"cluster_count",cluster_count)
            reference_network_filename = ref_path  + str(x)
            reference_network_filename = reference_network_filename.split("_" + author_type
+ "_MAIN.xml")[0] + '_' + author_type + '_REF.xml'
            reference_network = DataSet(reference_network_filename)
            #reference_network.reader_arnetminer(title_abstract_dict_original)
            reference_network.reader_arnetminer()
            #Undertaking inexact graph edit distance against each cluster for most
representative cluster for disambiugated author:
            print(set(predict_label_dict_fin.keys()))
            if (network_comparison == True) and len(predict_label_dict_fin) > 1 and ((not
set(predict_label_dict_fin.keys()) == set([0,-1])) or (not
set(predict_label_dict_fin.keys()) == set([-1,0]))):
                disambiguated_authors += 1
                for i in predict_label_dict_fin:
                    cluster = DataSet(filename,
predicted_cluster_dict=predict_label_dict_fin, cluster_index=i)
                    cluster.reader_arnetminer()
                    #cluster.reader_arnetminer(title_abstract_dict_original)
                    inexact_graph_edit_dist_cluster_dict[i] = []
                  #  print("AT inexact_graph_edit_dist_cluster_dict[i] = []")
                  #  print(i, "i")
                    print(len(predict_label_dict_fin[i]))
```

_____

BEGINNING STAGE THREE: RESEARCHER CONSOLIDATION FROM RESEARCHER DIAMBIGUATION OUTPUTS. FINDING DISAMBIGUATED CLUSTER WHICH IS MOST SIMILAR TO REFERENCE ACCOUNT OF RESEARCHER PROFILE FROM ORGANISATION USING GED.

_____

```
inexact_graph_edit_dist_cluster_dict[i].append(graph_edit_dist.compare(reference_network.C_
Graph,cluster.C_Graph))

inexact_graph_edit_dist_cluster_dict[i].append(graph_edit_dist.compare(reference_network.T_
Graph,cluster.T_Graph))

inexact_graph_edit_dist_cluster_dict[i].append(graph_edit_dist.compare(reference_network.Jc
onf_Graph,cluster.Jconf_Graph))
    #Year graph instead of org graph as no ref orgs -
inexact_graph_edit_dist_cluster_dict[i].append(graph_edit_dist.compare(reference_network.Or
g_Graph,cluster.Org_Graph))

inexact_graph_edit_dist_cluster_dict[i].append(graph_edit_dist.compare(reference_network.Ye
ar_Graph,cluster.Year_Graph))

inexact_graph_edit_dist_cluster_dict[i].append(graph_edit_dist.compare(reference_network.Ab
stract_Graph,cluster.Abstract_Graph))
                    print("Cluster GIED list created:",
inexact_graph_edit_dist_cluster_dict[i])
                    print("FINAL inexact_graph_edit_dist_cluster_dict is!",
inexact_graph_edit_dist_cluster_dict)
                    GED_cluster_avgs = [(i,mean(inexact_graph_edit_dist_cluster_dict[i]))
for i in inexact_graph_edit_dist_cluster_dict]
                    print("Average GED for clusters is",GED_cluster_avgs)
                    best_average_GED_cluster = [999,999]
                    for cluster in inexact_graph_edit_dist_cluster_dict:
                        clust_avg = mean(inexact_graph_edit_dist_cluster_dict[cluster])
                        print("Cluster",cluster, "has average GED of",clust_avg)
                        if clust_avg < best_average_GED_cluster[1]:
                            best_average_GED_cluster = [cluster,clust_avg]
                            print("New best_average_GED_cluster
is:",best_average_GED_cluster )
                final_cluster_index = best_average_GED_cluster[0]
                print("final cluster for author is",final_cluster_index)
                final_count,disambiguated_author_profile=
disambiguated_author_XMLconvertion(filename,file_path, x,ego,predict_label_dict_fin,
final_cluster_index)
                disambiguated_authors += 1
                disambiguated_author_profile.to_excel("./DISAMBIGUATED_AUTHOR_PROFILES/"+
ego + "_AUTHOR_DISAMBIG_PROFILE.xls")
                #adding status to summary dataframe:
                author_summary_dataframe.append([ego,"Disambiguated (k
clusters)",main_counts,ref_counts,cluster_count,
final_count,disambig_alg,place_holder,"None"])

            else:
```

```
                cluster_size_dict = dict()
                for index in set(predict_label_dict_fin.keys()):
                    cluster_size_dict[index] = len(predict_label_dict_fin[index])
                max_size = 0
                biggest_cluster = 0
                for cluster in cluster_size_dict.keys():
                    print(cluster)
                    if cluster_size_dict[cluster] > max_size:
                        print("cluster", cluster, "bigger then max_size:", max_size, "with
length of", cluster_size_dict[cluster])
                        biggest_cluster = cluster
                        max_size = cluster_size_dict[cluster]
                    elif cluster_size_dict[cluster] >= max_size:
                final_cluster_index = biggest_cluster
                # only 2 clusters or not wanting to use timely network comparison
                #non-optimized method, likely 2nd cluster is very small e.g. 2 papers -
with
                #default being to choose the biggest cluster from DBSCAN as author
representation and one most likely to be closest
                #to reference author dataset.
                print("Only one cluster for author, cluster index is", final_cluster_index)
                print("Final/biggest cluster is", final_cluster_index, "with paper
indexes:",predict_label_dict_fin[final_cluster_index] )
                #Now saving cluster index papers for primary author to XML file:
                final_count, disambiguated_author_profile=
disambiguated_author_XMLconvertion(filename,file_path, x,ego,predict_label_dict_fin,
final_cluster_index)
                disambiguated_authors += 1
                print("SAVING DISAMBIGUATED AUTHOR PROFILE EXCEL")
                disambiguated_author_profile.to_excel("./DISAMBIGUATED_AUTHOR_PROFILES/"+
ego + "_AUTHOR_DISAMBIG_PROFILE.xls")
                #adding status to summary dataframe
                author_summary_dataframe.append([ego,"Disambiguated (1
cluster)",main_counts,ref_counts,cluster_count,
final_count,disambig_alg,place_holder,"None"])
        else:
            print("Failed disambig comparison requirements defaulting to reference
network:", "main count:",main_counts,"ref_count:",ref_counts,"cluster_count",cluster_count)
            if ref_counts > 0: #author has reference papers
                #Then take reference network as XML authors publciations.
                final_count = default_XMLconvertion(file_path,x,ref_path,author_type,ego)
                defaulted_reference_authors += 1
                #adding status to summary dataframe
                author_summary_dataframe.append([ego,"Default Reference
Network",main_counts,ref_counts,cluster_count, final_count, "None",place_holder,"None"])
            else: #author has no reference papers
                print("Author failed disambiguation/network comparison and also failed
default to reference network")
                print("Likely main count and ref count are both 0:
Main:",main_counts,"Ref:",ref_counts)
                #add status to summary dataframe
```

```
author_summary_dataframe.append([ego,"Failed",main_counts,ref_counts,cluster_count,
final_count,"None",place_holder,"None"])
                failed_authors += 1
                '{} {}'.format(1, 2)
    author_summary_dataframe = pd.DataFrame(author_summary_dataframe, columns = ['Name',
'Status', 'Main Count','Reference Count', 'Cluster Count', 'Final_count',"Unsupervised
Clustering Method",'Read_Count',"Read_status"])
    return author_summary_dataframe, cnt, failed_authors, defaulted_reference_authors,
disambiguated_authors


            #Concludes disambiguation and profile consolidation functions and main

#Saving Author disambiguation and network comparison summary CSV for primary authors
primary_path = "./PRIMARY_AUTHORS_XML/"
primary_ref = "./PRIMARY_AUTHORS_XML//REF/"
summary, cnt, failed_authors, defaulted_reference_authors, disambiguated_authors =
primary_or_secondary_main(file_path=primary_path,ref_path=primary_ref,
author_type="PRIMARY",network_comparison=False,affinity_prop=False)
 (summary.loc[0])
summary.to_csv("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/2-3 -
Disambiguation and Network Comparison (IGM)/Disambiguation_summary.csv")
```

9.4 Research Topic Expertise Representation

———

Stage four: Topic Modelling of Researcher Publication Topics for capturing researcher expertise. Currently this involves three python scripts in Python 3. There is a dependence on a pre-defined and published static dataset of PubMed cancer-related papers. However, for this to be a functional tool generalisable to any research organisation domain it would be required to add in the essential – 'researcher training keyword extraction algorithm' – for extracting publications representing researchers interests from test or 'unseen' or actual researchers publications. We intend to implement a parallel algorithm for extraction that takes keywords from researcher initially extracted papers from all four interfaces (ORCID, Scopus, WOS, PubMed) & then reruns a secondary extraction to pull a significant amount of n papers proportional to the primary researchers amount of actual papers. Thereby, pulling primarily the test or unseen or actual papers of researchers as a test dataset for a trained LDA topic model, and secondarily then extracting/pooling a proportionally larger training dataset for training an LDA on publications that are unseen for researchers, though that pulls a corpus capturing the different pathways of researcher topic interest through paper keywords that is significantly larger in sample size for allowing for optimization of a human-interpretable topic model.
This would be run following extraction on prompt by user, with out of these python & R scripts within an Rshiny interface then projecting the LDAviz browser for users to decide on topic name allocations and then interactively add these within the Rshiny app, automatically updating user-feedback.

Furthermore, a recommendation for future user could be having this Shiny Reticulate application stored/delivered in a cloud server, with each user-interaction – extraction, training, testing of a LDA model – iteratively retraining a growing the research interests modelled, improving the utility of the application. This would work similarly, with stage five, whereby research recommendations & the trained model could be iteratively batch trained to improve recommendations & expand the network captured between organisation users.

———

Script 4.1 Running Pre-training Program for LDA Gensim Model on fixed PUBMED dataset

———

```
"""
File for PRE-TRAINING for Latent Dirichlet allocation (LDA) classification of cancer
researcher interests

****BASED ON GENSIM LDA DOCUMENTATION***

Stage four.one requires addition of 'training keyword search publication extraction
algorithm' for building of training set automatically during researcher extraction.
Thereby, this code pipeline would then use this training dataset based off researcher
profiles instead of a fixed assumed field of researcher - e.g. cancer related-publications
as opposed to a spread of cancer - related, and other health related publications following
our primary researchers changing interest in research topics.
"""
```

———

```
#Importing Libraries:
import re
import os
import numpy as np
import pandas as pd
from pprint import pprint

# Gensim
```

```
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

# spacy for lemmatization
import spacy

# Plotting tools
import pyLDAvis
import pyLDAvis.gensim  # don't skip this
import matplotlib.pyplot as plt

# Enable logging for gensim - optional
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',
level=logging.ERROR)

import warnings
warnings.filterwarnings("ignore",category=DeprecationWarning)

from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])

#Importing Biomedical PUBMED publication dataset for training LDA model adding these to the
orgs title abstract collection:
def get_file_list(file_dir):
    file_list = []
    for root, dirs, files in os.walk(file_dir):
        file_list.append(files)
    return file_list[0]
filedir = 'C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic
Representation/1-Secondary Author Pruning & Topic Modelling/cancer_PUBMED_dataset/'
PUBMED_papers = get_file_list(filedir)
paper_cnt = 0
PUBMED_title_abstract_dict = dict()
for file in PUBMED_papers:
    print("...",paper_cnt, "of", len(PUBMED_papers), "PUBMED papers remaining...")
    filename = filedir + file
    #print(filename)
    publication_data =  open(filename,"r",encoding='utf-8')
    pub_list = []
    cnt = 0
    for line in publication_data.readlines():
        if cnt % 2 != 0:
            pub_list.append(line)
            #print(line)
        cnt += 1
        # first index == title, second index == abstract, third index == keywords
        #print(pub_list)
```

```
    PUBMED_title_abstract_dict[pub_list[0]] = pub_list[1]
    paper_cnt += 1
print(paper_cnt, "files added to the PUBMED title_abstract_dict from PUBMED Biomedical and
Health Science Dataset as a training dataset for LDA.")

#Cleaning training dataset - based on PUBMED Cancer-related papers:
title_abstract_text_list = []
count = 0
for i in PUBMED_title_abstract_dict:
    count += 1
    abstract = PUBMED_title_abstract_dict[i]
    #adding in title with abstract:
    abstract = abstract + " " + i
    title_abstract_text_list.append(abstract)
data = title_abstract_text_list
print("EXTRACTED", len(title_abstract_text_list),"PUBLICATIONS FROM PUBMED-TRAINING TITLE-
ABSTRACT DICT")
# Remove new line characters
data = [re.sub('\s+', ' ', sent) for sent in data]

# Remove distracting single quotes
data = [re.sub("\'", "", sent) for sent in data]

def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))  # deacc=True
removes punctuations

data_words = list(sent_to_words(data))

# Build the bigram and trigram models
bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100) # higher threshold
fewer phrases.

# Faster way to get a sentence clubbed as a trigram/bigram
bigram_mod = gensim.models.phrases.Phraser(bigram)

# Initialize spacy 'en' model, keeping only tagger component (for efficiency)
#python3 -m spacy download en
nlp = spacy.load('en', disable=['parser', 'ner'])

# Define functions for stopwords, bigrams, trigrams and lemmatization
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words] for
doc in texts]

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    texts_out = []
```

```
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out

# Remove Stop Words
data_words_nostops = remove_stopwords(data_words)

# Form Bigrams
data_words_bigrams = make_bigrams(data_words_nostops)

# Do lemmatization keeping only noun, adj, vb, adv
data_lemmatized = lemmatization(data_words_bigrams, allowed_postags=['NOUN', 'ADJ', 'VERB',
'ADV'])

# Create Dictionary
id2word = corpora.Dictionary(data_lemmatized)

# Create Corpus
texts = data_lemmatized

# Term Document Frequency
corpus = [id2word.doc2bow(text) for text in texts]

def compute_coherence_values(corpus, dictionary, texts,limit, start=2, step=3):
    """
    Compute c_v coherence for various number of topics

    Parameters:
    ----------
    dictionary : Gensim dictionary
    corpus : Gensim corpus
    texts : List of input texts
    limit : Max num of topics

    Returns:
    -------
    model_list : List of LDA topic models
    coherence_values : Coherence values corresponding to the LDA model with respective
number of topics
    """
    coherence_values = []
    perplexity_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        print("Training LDA for", num_topics, "topics")
        model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                                id2word=id2word,
                                                num_topics=num_topics,
                                                random_state=100,
                                                update_every=1,
```

```
                                            chunksize=100,
                                            passes=10,
                                            alpha='auto',
                                            per_word_topics=True)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary,
coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())
        perplexity_values.append(model.log_perplexity(corpus))
    return model_list, coherence_values, perplexity_values
model_list, coherence_values,perplexity_values =
compute_coherence_values(dictionary=id2word, corpus=corpus, texts=data_lemmatized, start=1,
limit=51, step=5)

# Producing global Coherence graph output:
x = range(1, 51, 5)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.savefig("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/Evaluation Results
and Tables/E4-Topic Figures/Global_Coherence")
plt.show()

#Producing global Perplexity Graph Output:
x = range(1, 51, 5)
plt.plot(x, perplexity_values)
plt.xlabel("Num Topics")
plt.ylabel("Perplexity score")
plt.legend(("Perplexity_values"), loc='best')
plt.savefig("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/Evaluation Results
and Tables/E4-Topic Figures/Global_Perplexity")
plt.show()

# Print the coherence scores
print("FIRST GLOBAL RUN:")
for m, cv, pv in zip(x, coherence_values,perplexity_values):
    print("Num Topics =", m, " has Coherence Value of", round(cv, 4), "and Perplexity Value
of",round(pv,4))

limit=51; start=1; step=5;
topic_range = [num for num in range(start, limit, step)]
#Selecting best model to retrain for local topic range:
max_coherence = 0
best_model = ''
topic_num_final = 0
for model, coherence, perplexity, topic_num in zip(model_list,
coherence_values,perplexity_values, topic_range):
    if coherence > max_coherence:
        max_coherence = coherence
        best_model = model
```

```
        topic_num_final = topic_num
if topic_num_final - 5 < 1:
    topic_num_start = 1
elif not topic_num_final - 5 < 1:
    topic_num_start = topic_num_final - 5

#retraining for best model:
model_list, coherence_values,perplexity_values =
compute_coherence_values(dictionary=id2word, corpus=corpus, texts=data_lemmatized,
start=topic_num_start, limit=topic_num_final + 5, step=1)

# Print the coherence scores
print("SECOND LOCAL RUN:")
for m, cv, pv in zip(x, coherence_values,perplexity_values):
    print("Num Topics =", m, " has Coherence Value of", round(cv, 4), "and Perplexity Value
of",round(pv,4))

# Saving Local Coherence graph:
limit=topic_num_final + 5; start=topic_num_start; step=1;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.savefig("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/Evaluation Results
and Tables/E4-Topic Figures/Local_range_Coherence")
plt.show()

#Saving local Perplexity Graph:
limit=topic_num_final + 5; start=topic_num_start; step=1;
x = range(start, limit, step)
plt.plot(x, perplexity_values)
plt.xlabel("Num Topics")
plt.ylabel("Perplexity score")
plt.legend(("Perplexity_values"), loc='best')
plt.savefig("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/Evaluation Results
and Tables/E4-Topic Figures/Local_range_Perplexity")
plt.show()

topic_range = [num for num in range(start, limit, step)]
max_coherence = 0
best_model = ''
topic_num_final = 0
for model, coherence, perplexity, topic_num in zip(model_list,
coherence_values,perplexity_values, topic_range):
    if coherence > max_coherence:
        max_coherence = coherence
        best_model = model
        topic_num_final = topic_num
```

```
# Saving visualization of topics as html for users to vizualise LDA model topic distributions:
vis = pyLDAvis.gensim.prepare(best_model, corpus, id2word)
pyLDAvis.save_html(vis, 'C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic Representation/2-Assignment of Topic Labels using Modelled Topics/trained_LDA_topics.html')

# Saving model to disk for later import by main stage four topic labelling/allocation of primary researchers actual 'test' or unseen publciations.
from gensim.test.utils import datapath
temp_file = datapath("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic Representation/1-Secondary Author Pruning & Topic Modelling/Pre-trained_LDA_model/trained_LDA")
best_model.save(temp_file)
```

Script 4.2 Secondary Researcher (Co-author) Pruning for removed disambiguated publications & Topic Modelling

Context:

Secondary Pruning Protocol for Secondary Authors whose papers were removed in primary author disambiguation and further only taking secondary author papers which are co-authored with primary authors:

Secondary Author Pruning Protocol and Final disambiguated Dataset Construction:

1. Will extract a dataset from all disambig primary author XMLS

2. Create a dictionary of title: primary author list

3. Read in secondary author papers if paper title is in title: primary author list dictionary

4. If after reading secondary authors papers, secondary author no longer has any papers then remove the author from the final dataset

Following Author pruning, pre-trained LDA topic model imported and used to infer topics of all cancer researchers.

Then using topic vectors for each researcher over their set of papers, we calculate their topic vector centroids and use this to visualise their topic expertise network based on cosine similarity matrix between researchers centroids (using this as edge weight for subsequent visualisations - instead of individual edge topic links)

```
import pickle
import pandas as pd
import os
#Getting primary author names:
primary_authors = pd.read_csv("C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/2-3 - Disambiguation and Network
Comparison (IGM)/Disambiguation_summary.csv")
primary_authors_set = set(primary_authors.Name)

def get_title_abstract_dict():
    file_title_abstract = open('C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/2-3 - Disambiguation and Network
Comparison (IGM)/2-Running Disambiugation and Cluster
Comparison/PRIMARY_AUTHORS_XML/title_abstract_disseration_test.txt','r',encoding="
utf-8", errors='replace')
    title_abstract_dict = {}
    for line in file_title_abstract.readlines():
        #Getting document of titles and abstracts divided by <>
        if "<>" in line:
            arr=line.split("<>")
            #splits by key and value - title/abstract - returning as [title -
arr[0], abstract arr[1]]
            #print(line)
            #print(arr)
            if len(arr)>1:
                title_abstract_dict[arr[0].strip()]=arr[1].strip()
                #print("arr length > 1", title_abstract_dict[arr[0]])
                #print("Key", arr[0])
            else:
```

```
                title_abstract_dict[arr[0].strip()]=""
            # if length of arr is not > 1 then only a title is present without
abstract.
    return title_abstract_dict

title_abstract_dict = get_title_abstract_dict()

def get_file_list(file_dir):
    file_list = []
    for root, dirs, files in os.walk(file_dir):
        file_list.append(files)
    return file_list[0]

#Schema for primary dataset:
def primary_assemble(file_path,author_type):
    author_list = []
    file_path_list = file_path  + "disambiguated_main/"
    print(file_path_list)
    file_list = get_file_list(file_path_list)

    #Sorting file list alphabetically
    file_list = sorted(file_list)
    #Unsure of exact purpose
    file_list = file_list[:]
    #counter
    cnt = 0
    summary_final_primary_dict = {}
    #Begins iteration for each author file going into main function.
    #Extracting primary authors disambiguated files:
    for x in file_list:
        cnt += 1
        pub_cnt = 0
        read_status = "False"
        filename = file_path + "disambiguated_main/" + str(x)

        with open(filename, "r",encoding='utf-8') as filetoread:
            # 这里是一行一行读取的文件
            for line in filetoread:
                line = line.strip()
                #print(line)
                if "FullName" in line:
                    name = line[line.find('>')+1: line.rfind('<')].strip()
                elif "<title>" in line:
                    title = line[line.find('>')+1: line.rfind('<')].strip()
                elif "<DOI>" in line:
                    DOI = line[line.find('>')+1: line.rfind('<')].strip()
                elif "<jconf>" in line:
                    jconf = line[line.find('>')+1: line.rfind('<')].strip()
                elif "<year>" in line:
                    year = line[line.find('>')+1: line.rfind('<')].strip()
                elif "<organization>" in line:
```

```
                    organization = line[line.find('>')+1: line.rfind('<')].strip()
                elif "<authors>" in line:
                    authors = line[line.find('>')+1: line.rfind('<')].strip()
 #                elif ("<label>" in line):
 #                elif ("<year>" in line):  NOTE: NOT INCLUDED IN CURRENT METHOD.
                elif ("<total_cnt>" in line):
                    total_cnt = line[line.find('>')+1: line.rfind('<')].strip()
                elif "</publication>" in line:
                    topic_place_holder = ''
                    topic_place_holder2 = ''
                    abstract_place_holder = ''
                    author_list.append([year,name,
title,jconf,organization,authors,DOI,abstract_place_holder])
                    pub_cnt += 1
        print("Extracted", pub_cnt, "publications of", total_cnt, "total XML
publications for author", str(x))
        read_status = "True"
        summary_final_primary_dict[name] = [pub_cnt,read_status]
    print("Read in",cnt, "primary authors to final dataset")
    return author_list,summary_final_primary_dict,cnt

def secondary_assemble(file_path,author_type):
    author_list = []
    file_path_list = file_path  + "MAIN/"
    print(file_path_list)
    file_list = get_file_list(file_path_list)

    #Sorting file list alphabetically
    file_list = sorted(file_list)
    #Unsure of exact purpose
    file_list = file_list[:]
    print("File list is",file_list)
    #counter
    cnt = 0
    #Begins iteration for each author file going into main function.
    #Extracting primary authors disambiguated files:
    for x in file_list:
        cnt+=1
        pub_cnt = 0
        pruned_cnt = 0
        filename = file_path + "MAIN/" + str(x)

        with open(filename, "r",encoding='utf-8') as filetoread:
            # 这里是一行一行读取的文件
            for line in filetoread:
                line = line.strip()
                #print(line)
                if "FullName" in line:
                    name = line[line.find('>')+1: line.rfind('<')].strip()
                elif "<title>" in line:
                    title = line[line.find('>')+1: line.rfind('<')].strip()
```

```
                elif "<jconf>" in line:
                    jconf = line[line.find('>')+1: line.rfind('<')].strip()
                elif "<year>" in line:
                    year = line[line.find('>')+1: line.rfind('<')].strip()
                elif "<organization>" in line:
                    organization = line[line.find('>')+1: line.rfind('<')].strip()
                elif "<authors>" in line:
                    authors = line[line.find('>')+1: line.rfind('<')].strip()
                elif "<DOI>" in line:
                    DOI = line[line.find('>')+1: line.rfind('<')].strip()
 #                elif ("<label>" in line):
                elif ("<total_cnt>" in line):
                    total_cnt = line[line.find('>')+1: line.rfind('<')].strip()
                elif "</publication>" in line:
                    if (title in title_set) or (DOI in doi_set):
                        topic_place_holder = ''
                        topic_place_holder2 = ''
                        abstract_place_holder = ''
                        author_list.append([year, name,
title,jconf,organization,authors,DOI,abstract_place_holder])
                        pub_cnt += 1
                    else:
                        print("publication", DOI, "not in primary dataset pruning
secondary author publciation")
                        pruned_cnt += 1

        print("Extracted", pub_cnt, "publications of", total_cnt, "total XML
publications for author", str(x))
        print(pruned_cnt, "papers were pruned of a total",total_cnt,"for author",
str(x))
    print("Read in", cnt, "secondary authors to final dataset")
    return author_list, cnt

#Beginning primary and secondary author extraction for final dataset:
primary_disamb_path = "C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/2-3 - Disambiguation and Network
Comparison (IGM)/2-Running Disambiugation and Cluster
Comparison/PRIMARY_AUTHORS_XML/"

Primary_list,summary_primary_dict,primary_cnt =
primary_assemble(file_path=primary_disamb_path,author_type="PRIMARY")
Primary_dataframe = pd.DataFrame(Primary_list, columns = ['year','name', 'title',
'jconf','organization', 'coauthors',"DOI", "TOPICS"])
title_set = set(Primary_dataframe['title'])
doi_set = set(Primary_dataframe['DOI'])
disambig_summary = pd.read_csv("C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/2-3 - Disambiguation and Network
Comparison (IGM)/Disambiguation_summary.csv")
for i in range(len(disambig_summary)):
    if disambig_summary['Name'][i] in summary_primary_dict:
```

```
        disambig_summary['Read_Count'][i] =
summary_primary_dict[disambig_summary['Name'][i]][0]
        disambig_summary['Read_status'][i] =
summary_primary_dict[disambig_summary['Name'][i]][1]
disambig_summary.to_csv("C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/2-3 - Disambiguation and Network
Comparison (IGM)/Disambiguation_summary.csv")
Secondary_path = "C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/2-3 -
Disambiguation and Network Comparison (IGM)/2-Running Disambiugation and Cluster
Comparison/SECONDARY_AUTHORS_XML/"
Secondary_list, secondary_cnt =
secondary_assemble(file_path=Secondary_path,author_type="SECONDARY")
Secondary_dataframe = pd.DataFrame(Secondary_list, columns = ['year','name',
'title', 'jconf','organization', 'coauthors',"DOI","TOPICS"])
#ADD dois to XML extractions
#For secondary author read lines
#Iff paper title OR DOI IN primary_Dataset then append to secondary dataset- else
SKIP

#Combine primary and secondary datasets into final disambiguated dataset.
final_dataset = pd.concat([Primary_dataframe,
Secondary_dataframe],ignore_index=True )
print("Final dataset created with:",primary_cnt,"primary authors read
and",secondary_cnt,"secondary authors read into final dataset")

#Loading TRAINED GENSIM-LDA MODEL: Trained on 10000 PUBMED data mined/crawled
Cancer-related publications (title and abstract):
from gensim.test.utils import datapath
import gensim

# Loading pretrained LDA model from disk.
temp_file = datapath("C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic Representation/1-Secondary
Author Pruning & Topic Modelling/Pre-trained_LDA_model/trained_LDA")
trained_LDA = gensim.models.ldamodel.LdaModel.load(temp_file)

#Beginning text cleaning pipeline for primary organisations researchers papers:

import re
import numpy as np
import pandas as pd

# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess

# spacy for lemmatization
import spacy

# Enable logging for gensim - optional
```

```
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',
level=logging.ERROR)

import warnings
warnings.filterwarnings("ignore",category=DeprecationWarning)

from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])

def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))  #
deacc=True removes punctuations

def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in
stop_words] for doc in texts]

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in
allowed_postags])
    return texts_out

#Cleaning training dataset - based on PUBMED Cancer-related papers:
title_abstract_text_list = []
count = 0
for i in title_abstract_dict:
    count += 1
    abstract = title_abstract_dict[i]
    #adding in title with abstract:
    abstract = abstract + " " + i
    title_abstract_text_list.append(abstract)
# Remove new line characters
abstract = [re.sub('\s+', ' ', sent) for sent in title_abstract_text_list]

# Remove distracting single quotes
abstract = [re.sub("\'", "", sent) for sent in title_abstract_text_list]
abstract_words = list(sent_to_words(title_abstract_text_list))

# Build the bigram and trigram models
bigram = gensim.models.Phrases(abstract_words, min_count=5, threshold=100) #
higher threshold fewer phrases.
```

```
# Faster way to get a sentence clubbed as a trigram/bigram
bigram_mod = gensim.models.phrases.Phraser(bigram)

# Initialize spacy 'en' model, keeping only tagger component (for efficiency)
#python3 -m spacy download en
nlp = spacy.load('en', disable=['parser', 'ner'])

# Remove Stop Words
abstract_words_nostops = remove_stopwords(abstract_words)

# Form Bigrams
abstract_words_bigrams = make_bigrams(abstract_words_nostops)

# Lemmatization keeping only noun, adj, vb, adv
data_lemmatized = lemmatization(abstract_words_bigrams, allowed_postags=['NOUN',
'ADJ', 'VERB', 'ADV'])

# Creating wordid Dictionary
id2word = corpora.Dictionary(data_lemmatized)

#Creating inverse value-key dict for later calculation of topic researcher
centroids - note: no longer used.
id2word_inverse_dict = dict()
for key, value in zip(id2word.keys(), id2word.values()):
     id2word_inverse_dict[value] = key

# Creating Corpus - Term Document Frequency
new_corpus = [id2word.doc2bow(text) for text in data_lemmatized]

#Creating term-document frequency dictionary for later calculating researcher
centroids:
doc_corpus_term_frequency = dict()
for title, corpus in zip(title_abstract_dict.keys(),new_corpus ):
    doc_corpus_term_frequency[title] = corpus

title_topic_dict = dict()

#Beginning inference phase for adding topic probabilities for each researchers
unseen (test) publications:
for title,doc in zip(title_abstract_dict,new_corpus):
    vector = trained_LDA[doc]
    title_topic_dict[title] = dict()
    #for topic in vector:
    title_topic_dict[title] = vector[0]


#Creating set containing all the non-zero probability word ids for all words
associated with all topic distributions:
wordid_set = set()
topic_str = trained_LDA.show_topics(-1, len(id2word))
```

```
for topic, topstr in topic_str:
    topre = re.findall("(\d*\d.\d\d\d)\*\"(\w+)\"", topstr)
    for prob, word in topre:
        if float(prob) > 0.000:
            if word in id2word_inverse_dict:
                wordid_set.add(id2word_inverse_dict[word])

#Creating dictionary for each researcher, capturing the word tf (raw term
frequency) across all
#topic distributions of the total corpus. Using this as a way of calculating the
topic centroid for each
#author over these topic word distributions. Summing each tf for all researchers
papers and dividing each by
#the number of abstracts -> NOTE: TOPIC CENTROIDS NO LONGER USED.
researcher_topic_centroid_dict = dict()
researcher_topics_vector_centroid_dict = dict()
#Using ALL authors - primary author dataset to add topic labels for each author
#Creating an expertise dictionary for authors:
researcher_topic_representation_dict = dict()
author_paperscnt_dict = {}
author_expertise_dict = {}
author_topiccnt_dict = {}
unique_authors = set(final_dataset.name)
count = 0
for author in unique_authors:
    count += 1
    topic_set = set()
    topic_prop =set()
    print("...Beginning author {} - {} of {}".format(author, count,
len(unique_authors)))
    researcher_topic_representation_dict[author] = dict()
    researcher_topic_centroid_dict[author] = dict()
    #Creating a topic word distribution vector over all words with a default count
of zero for each author
    for wordid in wordid_set:
        researcher_topic_centroid_dict[author][wordid] = 0
    for num in range(len(wordid_set)): #number of topics from topic model
        researcher_topic_representation_dict[author][num] = 0
    paper_count = 0
    for row in range(len(final_dataset)):
        if final_dataset.iloc[row]['name'] == author:
            paper_count += 1
            instance = final_dataset.iloc[row]
            if instance['title'] in doc_corpus_term_frequency:
                for wordid, freq in doc_corpus_term_frequency[title]:
                    if wordid in researcher_topic_centroid_dict[author]:
                        researcher_topic_centroid_dict[author][wordid] +=freq
                    if not wordid in researcher_topic_centroid_dict[author]:
                        researcher_topic_centroid_dict[author][wordid] = freq
                for topic, proportion in title_topic_dict[instance['title']]:
```

```
                    researcher_topic_representation_dict[author][topic] +=
proportion
                    topic_set.add(topic)

    author_paperscnt_dict[author] = paper_count
    author_topiccnt_dict[author] = topic_set
    #Now dividing each topic word frequency count by the number of papers and
adding to vector of
    ##each researchers topic word centroid or topic representation:
    researcher_topic_vector = []
    for wordindex in researcher_topic_centroid_dict[author]:
        researcher_topic_centroid_dict[author][wordid] =
int(researcher_topic_centroid_dict[author][wordid]/ author_paperscnt_dict[author])

researcher_topic_vector.append(researcher_topic_centroid_dict[author][wordid])
        researcher_topics_vector_centroid_dict[author] =
np.array(researcher_topic_vector)

    for topic in researcher_topic_representation_dict[author]:
        #Creating final topic proportions for author:
        researcher_topic_representation_dict[author][topic] =
researcher_topic_representation_dict[author][topic]/author_paperscnt_dict[author]
    author_expertise_dict[author] = researcher_topic_representation_dict[author]
#Creating the cosine similarity matrix for all researchers based on their topic
representation vectors:
#import pandas as pd#
#COSINE SIMILARITY MATRIX NO LONGER USED FOR CALCULATING TOPIC CENTROID/TOPIC
SIMILARITY
#import math
#def cosine_similarity(v1,v2):
#  # "compute cosine similarity of v1 to v2: (v1 dot v2)/{||v1||*||v2||)"
#    sumxx, sumxy, sumyy = 0, 0, 0
#    for i in range(len(v1)):
#        x = v1[i]; y = v2[i]
#        sumxx += x*x
#        sumyy += y*y
#        sumxy += x*y
#    return sumxy/math.sqrt(sumxx*sumyy)

#Creating dictionary for sum of researchers topic vector counts -> NO LONGER USED.
#researcher_topics_vector_truth_dict = dict()
#for researchers in researcher_topics_vector_centroid_dict:
#    running_total = 0.0
#    for count in researcher_topics_vector_centroid_dict[researchers]:
#        running_total += count
#    if running_total == 0:
#       researcher_topics_vector_truth_dict[researchers] = False
#researchers =[ researchers for researchers in
researcher_topics_vector_centroid_dict.keys() if not researchers in
researcher_topics_vector_truth_dict and researchers in primary_authors_set]
```

```
#This needs to be optimized - very slow.  -> NO LONGER USED.
#author_pairs_dict_topic = dict()
#def cossim_matrix(researchers):
#    for i in range(len(researchers)):
#        for j in range(len(researchers)):
##            if researchers[i] != researchers[j]:
#                if not ((researchers[i],researchers[j])) in
author_pairs_dict_topic and not ((researchers[j],researchers[i])) in
author_pairs_dict_topic:
#                    author_pairs_dict_topic[((researchers[i],researchers[j]))] =
round(cosine_similarity(researcher_topics_vector_centroid_dict[researchers[i]],res
earcher_topics_vector_centroid_dict[researchers[j]]),2)
#cossim_matrix(researchers)

#pickling_on = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-
Topic Representation/1-Secondary Author Pruning & Topic
Modelling/Researcher_cosine_distance_dict.pickle","wb")
#pickle.dump(author_pairs_dict_topic, pickling_on)
#pickling_on.close()


for row in range(len(final_dataset)):
    if final_dataset.iloc[row]['title'] in title_abstract_dict:
        final_dataset.iloc[row]['abstract'] =
title_abstract_dict[final_dataset.iloc[row]['title']]

for row in range(len(final_dataset)):
    #if final_dataset.iloc[row]['title'] in title_abstract_dict:
       # final_dataset.iloc[row]['abstract'] =
title_abstract_dict[final_dataset.iloc[row]['title']]
    if final_dataset.iloc[row]['title'] in title_topic_dict:
       # final_dataset.iloc[row]['Primary_topic_labels'] =
author_expertise_dict_labels[final_dataset.iloc[row]['name']][0]
        #final_dataset.iloc[row]['Secondary_topic_labels'] =
author_expertise_dict_labels[final_dataset.iloc[row]['name']][1]
        topics_vec = []
        for topic, prop in title_topic_dict[final_dataset.iloc[row]['title']]:
            topics_vec.append((topic, prop))
        final_dataset.iloc[row]['TOPICS'] = topics_vec

#Creating a CSV file for capturing each researchers data:
dataset_list = []
for author in set(final_dataset['name']):
    Name = author
    Titles = set()
    Venues = set()
    Organizations = set()
    Coauthors = set()
    Number_Coauthors = -1 #not including author
    DOIs = set()
    if author in author_expertise_dict:
```

```
        Topics = author_expertise_dict[author]
    else:
        Topics = 'NA'
    if author in author_paperscnt_dict:
        Paper_number = author_paperscnt_dict[author]
    else:
        Paper_number = 'NA'
    Abstracts = set()
    for row in range(len(final_dataset)):
        if author == final_dataset.iloc[row]['name']:
            Titles.add(final_dataset.iloc[row]['title'])
            Venues.add(final_dataset.iloc[row]['jconf'])
            Organizations.add(final_dataset.iloc[row]['organization'])
            coauthor_list = final_dataset.iloc[row]['coauthors'].split(' ')
            for coauthor in coauthor_list:
                Coauthors.add(coauthor)
            DOIs.add(final_dataset.iloc[row]['DOI'])
    Number_Coauthors += len(Coauthors)

dataset_list.append([Name,Titles,Paper_number,Venues,Organizations,Coauthors,Numbe
r_Coauthors,DOIs,Topics])

All_authors_tpc_modelled = pd.DataFrame(dataset_list, columns = ['Name',
'Titles','No. Papers', 'Venues','Organizations', 'Coauthors',"No. Coauthors",
"DOIs","Topics"])
All_authors_tpc_modelled.to_csv("C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic Representation/1-Secondary
Author Pruning & Topic Modelling/final_dataset_TPC_Modelled_Summary.csv")

pickling_on = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-
Topic Representation/1-Secondary Author Pruning & Topic
Modelling/Final_dataset.pickle","wb")
pickle.dump(final_dataset, pickling_on)
pickling_on.close()

pickling_on = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-
Topic Representation/1-Secondary Author Pruning & Topic
Modelling/Final_dataset_author.pickle","wb")
pickle.dump(All_authors_tpc_modelled, pickling_on)
pickling_on.close()

#Creating excel for primary authors only:
#Creating a CSV file for capturing each researchers data:
primary_dataset_list = []
for author in set(final_dataset['name']):
    if author in primary_authors_set:
        Name = author
        Titles = set()
        Venues = set()
        Organizations = set()
        Coauthors = set()
```

```
        Number_Coauthors = -1 #not including author
        DOIs = set()
        if author in author_expertise_dict:
            Topics = author_expertise_dict[author]
        else:
            Topics = 'NA'
        if author in author_paperscnt_dict:
            Paper_number = author_paperscnt_dict[author]
        else:
            Paper_number = 'NA'
        Abstracts = set()
        for row in range(len(final_dataset)):
            if author == final_dataset.iloc[row]['name']:
                Titles.add(final_dataset.iloc[row]['title'])
                Venues.add(final_dataset.iloc[row]['jconf'])
                Organizations.add(final_dataset.iloc[row]['organization'])
                coauthor_list = final_dataset.iloc[row]['coauthors'].split(' ')
                for coauthor in coauthor_list:
                    Coauthors.add(coauthor)
                DOIs.add(final_dataset.iloc[row]['DOI'])
        Number_Coauthors += len(Coauthors)

primary_dataset_list.append([Name,Titles,Paper_number,Venues,Organizations,Coautho
rs,Number_Coauthors,DOIs,Topics])

Primary_tpc_modelled = pd.DataFrame(primary_dataset_list, columns = ['Name',
'Titles','No. Papers', 'Venues','Organizations', 'Coauthors',"No. Coauthors",
"DOIs","Topics"])
Primary_tpc_modelled.to_csv("C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic
Representation/Primary_Author_TOPIC_Summary.csv")
```

Script 4.3 Assignment of Topic Labels to Modelled Topics of Pre-trained LDA Language model

```
"""
Instructions:

1. To begin the assignment of topic labels, within the 2-Assignment of Topic Labels using Modelled Topics
   folder from 4-Topic Representation, open the html trained_LDA_topics file for a visualisation
   of the trained research topic distributions for your research organisation.

   Using this visualisation, run this python script and enter a topic label when requested for each topic where you are sure of the expertise label
   as inferred from the word distribution of the topic.

Currently a manual process, run through scripts. Would preferably end up being interactive run through an Rshiny & Reticulate interface between Python and R.
"""
```

```
import pickle
import networkx as nx
import pandas as pd
from jieba import analyse
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re

#Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic Representation/1-Secondary Author Pruning & Topic Modelling/Final_dataset.pickle",'rb')
#Final_dataset = pickle.load(file)
unpickler = pickle.Unpickler(file)
Author_dataset = unpickler.load()

#Loading in cleaned and topic-modelled researcher dataset by researcher paper sets:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic Representation/1-Secondary Author Pruning & Topic Modelling/Final_dataset_author.pickle",'rb')
#Final_dataset = pickle.load(file)
unpickler = pickle.Unpickler(file)
Author_dataset_RESEARCHER = unpickler.load()

#Collecting topics:
Topics = set()
for topic_list in Author_dataset['TOPICS'].values:
    for topic, prop in topic_list:
        Topics.add(topic)
Topics = list(Topics)
```

```
#Requesting user to add a new topic label for each topic where possible:
Inferred_topic_label_dict = dict()
print("Using the topic vizualization file add topic labels where confident in the topic
expertise")
for topic in Topics:
    print("\n")
    print("Topic is", str(int(topic) + 1))
    topic_request = input("Would you like to add a topic label to this topic? [yes|no]:\n")
    if topic_request.lower() == "yes":
        topic_label = input("Insert new topic label for topic distribution:\n")
    else:
        topic_label = "T" + str(int(topic) + 1)
    print("Topic Label is", topic_label)
    Inferred_topic_label_dict["T"+str(topic + 1)] = topic_label

pickling_on = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic
Representation/2-Assignment of Topic Labels using Modelled
Topics/Inferred_topic_label_dict.pickle","wb")
pickle.dump(Inferred_topic_label_dict, pickling_on)
pickling_on.close()
```

9.5 Research Collaboration Link Prediction & Recommendations

Heterogeneous Network Construction, Link Prediction & Recommendations using Python Scripts. As this leverages StellarGraph which can connect it's model directly to Neo4j and run off this graph database, in future during network construction these organisation networks could be iteratively added to a Neo4j database. Allowing for an ever-expanding research collaboration network and more effective research collaboration recommendations & visualisations. Furthermore, being stored in the cloud, during each users use the research collaboration network model and metapath2vec embeddings could be retrained or batch trained to improve during each user session, essentially network mapping regional, state and national research communities. A key adjustment for this section would be the need to incorporate pub year, country and other dimensions and most significantly turn the static graph used into a dynamic graph to allow for recommendation predictions that predict over time periods, thereby becoming closer to reality.

Script 5.1 Heterogeneous Network Construction: Preparation of Multi-type Nodes (Venue, Organisation, Topic & Researcher) & Multi-type edges (relationships) between nodes, pickling as dataframe for link prediction script.

```
#import libraries:
from stellargraph import StellarGraph
import pickle
import networkx as nx
import pandas as pd
from jieba import analyse
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re

#Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic
Representation/1-Secondary Author Pruning & Topic Modelling/Final_dataset.pickle",'rb')
#Final_dataset = pickle.load(file)
unpickler = pickle.Unpickler(file)
Author_dataset = unpickler.load()

#Loading in cleaned and topic-modelled researcher dataset by researcher paper sets:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic
Representation/1-Secondary Author Pruning & Topic
Modelling/Final_dataset_author.pickle",'rb')
#Final_dataset = pickle.load(file)
unpickler = pickle.Unpickler(file)
Author_dataset_RESEARCHER = unpickler.load()
unique_authors = set(Author_dataset.name)

#Extracting source-target edges for homogenous dataframes representations of PAPER-relation
network:
#edge_set_YEAR = set()
#edge_set_COUNTRY = set() NOTE: These two variables are not currently used.
edge_weight_dict = dict()
nodes_edge_list_of_lists = []
edge_set_author_P_ORG = set()
edge_set_author_P_author = set()
edge_set_author_P_venue = set()
edge_set_author_P_topic = set()
```

```
edge_set_org_A_org = set()
edge_set_venue_A_venue = set()
author_expertise_dict = {}
org_author_dict = {}
venue_author_dict = {}
venue_title_dict = {}
org_title_dict = {}
topic_author_dict = dict()
topic_title_dict = dict()

for index in range(len(Author_dataset)):
    #IN FUTURE ADDING IN YEAR FOR TRAIN/TEST/DEV SPLITS IN LINK PRED as temporal dataset:
    print("Beginning AUTHOR -> PAPER -> AUTHOR -relation extraction - {} of
{}".format(index, len(Author_dataset)))
    author1 = Author_dataset["name"][index]
    author1_topic = Author_dataset["TOPICS"][index]

    #Extracting author <(P)> author CO-AUTHOR edges:
    coauthor_list = Author_dataset["coauthors"][index].split(" ")
    for coauthor in coauthor_list:
        if not coauthor == author1:
            if (not tuple((author1,coauthor)) in edge_set_author_P_author) and (not
tuple((coauthor,author1)) in edge_set_author_P_author):
#As graph is multi-graph and undirected so only need to represent edge between two - this
would be a duplicate edge otherwise.
                edge_set_author_P_author.add(tuple((author1,coauthor)))
            if not (tuple((author1,coauthor)) in edge_weight_dict) and (not
tuple((coauthor,author1)) in edge_weight_dict):
                edge_weight_dict[tuple((author1,coauthor))] = {"AUTHOR-(P)-AUTHOR": 1}
            elif tuple((author1,coauthor)) in edge_weight_dict:
                if "AUTHOR-(P)-AUTHOR" in edge_weight_dict[tuple((author1,coauthor))]:
                    edge_weight_dict[tuple((author1,coauthor))]["AUTHOR-(P)-AUTHOR"] += 1
                else:
                    edge_weight_dict[tuple((author1,coauthor))]["AUTHOR-(P)-AUTHOR"] = 1
            elif tuple((coauthor,author1)) in edge_weight_dict:
                if "AUTHOR-(P)-AUTHOR" in edge_weight_dict[tuple((coauthor,author1))]:
                    edge_weight_dict[tuple((coauthor,author1))]["AUTHOR-(P)-AUTHOR"] += 1
                else:
                    edge_weight_dict[tuple((coauthor,author1))]["AUTHOR-(P)-AUTHOR"] = 1
            else:
                print("None (else) condition entered for coauthor-papers!")
        else:
            print("Skipping edge: Coauthor == Author")

    #Extracting AUTHOR <-> (PAPER) <-> ORGANISATION edges:   Belongs:  note: CO-ORG =
AUTHOR <-> ORG <-> AUTHOR
    org = Author_dataset["organization"][index]
    if org != "NA":
        if (not tuple((author1,org)) in edge_set_author_P_ORG): #As graph is multi-graph
and undirected so only need to represent edge between two - this would be a duplicate edge
otherwise.
```

```
            edge_set_author_P_ORG.add(tuple((author1,org)))
        if not (tuple((author1,org)) in edge_weight_dict):
            edge_weight_dict[tuple((author1,org))] = {"AUTHOR-(P)-ORG": 1}
        elif tuple((author1,org)) in edge_weight_dict:
            if "AUTHOR-(P)-ORG" in edge_weight_dict[tuple((author1,org))]:
                edge_weight_dict[tuple((author1,org))]["AUTHOR-(P)-ORG"] += 1
            else:
                edge_weight_dict[tuple((author1,org))]["AUTHOR-(P)-ORG"] = 1
        else:
            print("None (else) condition entered for AUTHOR-(P)-ORG!")

        #Building author organisation dictionary for later extraction of ORG <-> AUTHOR <-> ORG edges:
        if org != "NA":
            if org in org_author_dict:
                if author1 in org_author_dict[org]:
                    org_author_dict[org][author1] += 1
                elif author1 not in org_author_dict[org]:
                    org_author_dict[org][author1] = 1
            elif not org in org_author_dict:
                org_author_dict[org] = dict()
                org_author_dict[org][author1] = 1

    #Extracting AUTHOR <-> (PAPER) <-> VENUE edges:   Belongs:  note: CO-VENUE = AUTHOR <-> VENUE <-> AUTHOR
    venue = Author_dataset["jconf"][index]
    if venue != "NA":
        if (not tuple((author1,venue)) in edge_set_author_P_venue): #As graph is multi-graph and undirected so only need to represent edge between two - this would be a duplicate edge otherwise.
            edge_set_author_P_venue.add(tuple((author1,venue)))
        if not (tuple((author1,venue)) in edge_weight_dict):
            edge_weight_dict[tuple((author1,venue))] = {"AUTHOR-(P)-VENUE": 1}
        elif tuple((author1,venue)) in edge_weight_dict:
            if "AUTHOR-(P)-VENUE" in edge_weight_dict[tuple((author1,venue))]:
                edge_weight_dict[tuple((author1,venue))]["AUTHOR-(P)-VENUE"] += 1
            else:
                edge_weight_dict[tuple((author1,venue))]["AUTHOR-(P)-VENUE"] = 1
        else:
            print("None (else) condition entered for AUTHOR-(P)-VENUE!")


        #Building author-venue dictionary for later extraction of VENUE <-> AUTHOR <-> VENUE (co-publishers) edges:
        if venue != "NA":
            if venue in venue_author_dict:
                if author1 in venue_author_dict[venue]:
                    venue_author_dict[venue][author1] += 1
                elif author1 not in venue_author_dict[venue]:
                    venue_author_dict[venue][author1] = 1
            elif not venue in venue_author_dict:
```

```
                venue_author_dict[venue] = dict()
                venue_author_dict[venue][author1] = 1

    #Extracting AUTHOR <- (PAPER) -> TOPIC edges:
    topic_list = Author_dataset["TOPICS"][index]
    if topic_list != []:
        for topic, prop in topic_list:
            topic = "T{}".format(topic)
            if prop > 0.20:
                if (not tuple((author1,topic)) in edge_set_author_P_topic): #As graph is
multi-graph and undirected so only need to represent edge between two - this would be a
duplicate edge otherwise.
                    edge_set_author_P_topic.add(tuple((author1,topic)))
                if not (tuple((author1,topic)) in edge_weight_dict):
                    edge_weight_dict[tuple((author1,topic))] = {"AUTHOR-(P)-TOPIC":
round(prop,2)}
                elif tuple((author1,topic)) in edge_weight_dict:
                    if "AUTHOR-(P)-TOPIC" in edge_weight_dict[tuple((author1,topic))]:
                        edge_weight_dict[tuple((author1,topic))]["AUTHOR-(P)-TOPIC"] +=
round(prop,2)
                    else:
                        edge_weight_dict[tuple((author1,topic))]["AUTHOR-(P)-TOPIC"] =
round(prop,2)
                else:
                    print("None (else) condition entered for AUTHOR-(P)-TOPIC!")

        #Building author-topic dictionary for later extraction of TOPIC <-> AUTHOR <->
TOPIC (co-publishers) edges:
        if topic_list != []:
            for topic, prop in topic_list:
                if prop >0.20:
                    topic = "T{}".format(topic)
                    if topic in topic_author_dict:
                       if author1 in topic_author_dict[topic]:
                           topic_author_dict[topic][author1] += round(prop,2)
                       elif author1 not in topic_author_dict[topic]:
                           topic_author_dict[topic][author1] = round(prop,2)
                    elif not topic in topic_author_dict:
                        topic_author_dict[topic] = dict()
                        topic_author_dict[topic][author1] = round(prop,2)

#PAPER ONLY EDGES:
Paper_dataset = Author_dataset.iloc[:,1:]
Paper_dataset.drop_duplicates(subset=["title"])
for index in range(len(Paper_dataset)):
   #Extracting separate topic, organisation and venue title dictionaries for TOPIC <P> ORG,
TOPIC <P> VENUE and  VENUE <-> PAPER <-> ORGANISATION (co-publishers) edges:
    title = Author_dataset["title"][index]
    venue = Author_dataset["jconf"][index]
    org = Author_dataset["organization"][index]
    topic_list = Author_dataset["TOPICS"][index]
```

```
    if title != "NA" and venue != "NA":
        if venue in venue_title_dict:
            if title in venue_title_dict[venue]:
                venue_title_dict[venue][title] += 1
            elif title not in venue_title_dict[venue]:
                venue_title_dict[venue][title] = 1
        elif not venue in venue_title_dict:
            venue_title_dict[venue] = dict()
            venue_title_dict[venue][title] = 1
    if title != "NA" and org != "NA":
        if org in org_title_dict:
            if title in org_title_dict[org]:
                org_title_dict[org][title] += 1
            elif title not in org_title_dict[org]:
                org_title_dict[org][title] = 1
        elif not org in org_title_dict:
            org_title_dict[org] = dict()
            org_title_dict[org][title] = 1
    if topic_list != [] and title != "NA":
        for topic, prop in topic_list:
            topic = "T{}".format(topic)
            if topic in topic_title_dict:
                if title in topic_title_dict[topic]:
                    topic_title_dict[topic][title] += round(prop,2)
                elif title not in topic_title_dict[topic]:
                    topic_title_dict[topic][title] = round(prop,2)
            elif not topic in topic_title_dict:
                topic_title_dict[topic] = dict()
                topic_title_dict[topic][title] = round(prop,2)

#ORG <-> (PAPER) <-> VENUE Relation:
edge_set_org_P_venue = set()
for org in org_title_dict:
    for venue in venue_title_dict:
        if (org != "NA") and (venue != "NA") and (not tuple((org,venue)) in
edge_set_org_P_venue) and (not tuple((venue, org)) in edge_set_org_P_venue) and (not venue
== "NA") and (not org == "NA"):
            paper_intersection =
set(org_title_dict[org].keys()).intersection(set(venue_title_dict[venue].keys()))
            ven_org_edge_weight = len(paper_intersection)
            if ven_org_edge_weight != 0:
                if (not tuple((org,venue)) in edge_set_org_P_venue) and (not tuple((venue,
org)) in edge_set_org_P_venue):
#As graph is multi-graph and undirected so only need to represent edge between two - this
would be a duplicate edge otherwise.
                    edge_set_org_P_venue.add(tuple((org,venue)))
                if not (tuple((org,venue)) in edge_weight_dict) and (not tuple((venue,
org)) in edge_weight_dict):
                    edge_weight_dict[tuple((org,venue))] = {"ORG-(P)-VENUE":
ven_org_edge_weight}
                elif tuple((org,venue)) in edge_weight_dict:
```

```
                    if "ORG-(P)-VENUE" in edge_weight_dict[tuple((org,venue))]:
                        edge_weight_dict[tuple((org,venue))]["ORG-(P)-VENUE"] +=
ven_org_edge_weight
                    else:
                        edge_weight_dict[tuple((org,venue))]["ORG-(P)-VENUE"] =
ven_org_edge_weight
                elif tuple((venue,org)) in edge_weight_dict:
                    if "ORG-(P)-VENUE" in edge_weight_dict[tuple((venue,org))]:
                        edge_weight_dict[tuple((venue,org))]["ORG-(P)-VENUE"] +=
ven_org_edge_weight
                    else:
                        edge_weight_dict[tuple((venue,org))]["ORG-(P)-VENUE"] =
ven_org_edge_weight
                else:
                    print("None (else) condition entered for ORG-(P)-VENUE edge!")
#TOPIC <- (P) -> ORG
edge_set_org_P_topic = set()
for org in org_title_dict:
    for topic in topic_title_dict:
        if (org != "NA") and (topic != "NA") and (not tuple((org,topic)) in
edge_set_org_P_topic) and (not tuple((topic, org)) in edge_set_org_P_topic) and (not topic
== "NA") and (not org == "NA"):
            paper_intersection =
set(org_title_dict[org].keys()).intersection(set(topic_title_dict[topic].keys()))
            topic_org_edge_weight = len(paper_intersection)
            if topic_org_edge_weight != 0:
                if (not tuple((org,topic)) in edge_set_org_P_topic) and (not tuple((topic,
org)) in edge_set_org_P_topic): #As graph is multi-graph and undirected so only need to
represent edge between two - this would be a duplicate edge otherwise.
                    edge_set_org_P_topic.add(tuple((org,topic)))
                if not (tuple((org,topic)) in edge_weight_dict) and (not tuple((topic,
org)) in edge_weight_dict):
                    edge_weight_dict[tuple((org,topic))] = {"ORG-(P)-TOPIC":
topic_org_edge_weight}
                elif tuple((org,topic)) in edge_weight_dict:
                    if "ORG-(P)-TOPIC" in edge_weight_dict[tuple((org,topic))]:
                        edge_weight_dict[tuple((org,topic))]["ORG-(P)-TOPIC"] +=
topic_org_edge_weight
                    else:
                        edge_weight_dict[tuple((org,topic))]["ORG-(P)-TOPIC"] =
topic_org_edge_weight
                elif tuple((topic,org)) in edge_weight_dict:
                    if "ORG-(P)-TOPIC" in edge_weight_dict[tuple((topic,org))]:
                        edge_weight_dict[tuple((topic,org))]["ORG-(P)-TOPIC"] +=
topic_org_edge_weight
                    else:
                        edge_weight_dict[tuple((topic,org))]["ORG-(P)-TOPIC"] =
topic_org_edge_weight
                else:
                    print("None (else) condition entered for ORG-(P)-TOPIC edge!")
```

```
#TOPIC <- (PAPER) - > VENUE
edge_set_topic_P_venue = set()
for topic in topic_title_dict:
    for venue in venue_title_dict:
        print(venue)
        print(topic)
        if (topic != "NA") and (venue != "NA") and (not tuple((topic,venue)) in
edge_set_topic_P_venue) and (not tuple((venue, topic)) in edge_set_topic_P_venue) and (not
venue == "NA") and (not topic == "NA"):
            paper_intersection =
set(topic_title_dict[topic].keys()).intersection(set(venue_title_dict[venue].keys()))
            ven_topic_edge_weight = len(paper_intersection)
            if ven_topic_edge_weight != 0:
                if (not tuple((topic,venue)) in edge_set_topic_P_venue) and (not
tuple((venue, topic)) in edge_set_topic_P_venue): #As graph is multi-graph and undirected
so only need to represent edge between two - this would be a duplicate edge otherwise.
                    edge_set_topic_P_venue.add(tuple((topic,venue)))
                if not (tuple((topic,venue)) in edge_weight_dict) and (not tuple((venue,
topic)) in edge_weight_dict):
                    edge_weight_dict[tuple((topic,venue))] = {"TOPIC-(P)-VENUE":
ven_topic_edge_weight}
                elif tuple((topic,venue)) in edge_weight_dict:
                    if "TOPIC-(P)-VENUE" in edge_weight_dict[tuple((topic,venue))]:
                        edge_weight_dict[tuple((topic,venue))]["TOPIC-(P)-VENUE"] +=
ven_topic_edge_weight
                    else:
                        edge_weight_dict[tuple((topic,venue))]["TOPIC-(P)-VENUE"] =
ven_topic_edge_weight
                elif tuple((venue,topic)) in edge_weight_dict:
                    if "TOPIC-(P)-VENUE" in edge_weight_dict[tuple((venue,topic))]:
                        edge_weight_dict[tuple((venue,topic))]["TOPIC-(P)-VENUE"] +=
ven_topic_edge_weight
                    else:
                        edge_weight_dict[tuple((venue,topic))]["TOPIC-(P)-VENUE"] =
ven_topic_edge_weight
                else:
                    print("None (else) condition entered for TOPIC-(P)-VENUE edge!")

#ORG <-> (AUTHOR) <-> ORG Relation:  CO-STAFF-ORG:
for org1 in org_author_dict:
    for org2 in org_author_dict:
        if (org1 != "NA") and (org2 != "NA") and (org1 != org2) and (not tuple((org1,org2))
in edge_set_org_A_org) and (not tuple((org2,org1)) in edge_set_org_A_org):
            author_intersection =
set(org_author_dict[org1].keys()).intersection(set(org_author_dict[org2].keys()))
            org_edge_weight = len(author_intersection)
            if org_edge_weight != 0:
                if (not tuple((org1,org2)) in edge_set_org_A_org) and (not
tuple((org2,org1)) in edge_set_org_A_org): #As graph is multi-graph and undirected so only
need to represent edge between two - this would be a duplicate edge otherwise.
                    edge_set_org_A_org.add(tuple((org1,org2)))
```

```
                if not (tuple((org1,org2)) in edge_weight_dict) and (not tuple((org2,org1))
in edge_weight_dict):
                    edge_weight_dict[tuple((org1,org2))] = {"ORG-(A)-ORG": org_edge_weight}
                elif tuple((org1,org2)) in edge_weight_dict:
                    if "ORG-(A)-ORG" in edge_weight_dict[tuple((org1,org2))]:
                        edge_weight_dict[tuple((org1,org2))]["ORG-(A)-ORG"] +=
org_edge_weight
                    else:
                        edge_weight_dict[tuple((org1,org2))]["ORG-(A)-ORG"] =
org_edge_weight
                elif tuple((org2,org1)) in edge_weight_dict:
                    if "ORG-(A)-ORG" in edge_weight_dict[tuple((org2,org1))]:
                        edge_weight_dict[tuple((org2,org1))]["ORG-(A)-ORG"] +=
org_edge_weight
                    else:
                        edge_weight_dict[tuple((org2,org1))]["ORG-(A)-ORG"] =
org_edge_weight
                else:
                    print("None (else) condition entered for ORG-A_ORG edge!")

#VENUE <-> (AUTHOR) <-> VENUE Relation:  CO-PUB VENUE:
for venue1 in venue_author_dict:
    for venue2 in venue_author_dict:
        if (venue1 != "NA") and (venue2 != "NA") and (venue1 != venue2) and (not
tuple((venue1,venue2)) in edge_set_venue_A_venue) and (not tuple((venue2,venue1)) in
edge_set_venue_A_venue):
            author_intersection =
set(venue_author_dict[venue1].keys()).intersection(set(venue_author_dict[venue2].keys()))
            venue_edge_weight = len(author_intersection)
            if venue_edge_weight != 0:
                if (not tuple((venue1,venue2)) in edge_set_venue_A_venue) and (not
tuple((venue2,venue1)) in edge_set_venue_A_venue): #As graph is multi-graph and undirected
so only need to represent edge between two - this would be a duplicate edge otherwise.
                    edge_set_venue_A_venue.add(tuple((venue1,venue2)))
                if not (tuple((venue1,venue2)) in edge_weight_dict) and (not
tuple((venue2,venue1)) in edge_weight_dict):
                    edge_weight_dict[tuple((venue1,venue2))] = {"VENUE-(A)-VENUE":
venue_edge_weight}
                elif tuple((venue1,venue2)) in edge_weight_dict:
                    if "VENUE-(A)-VENUE" in edge_weight_dict[tuple((venue1,venue2))]:
                        edge_weight_dict[tuple((venue1,venue2))]["VENUE-(A)-VENUE"] +=
venue_edge_weight
                    else:
                        edge_weight_dict[tuple((venue1,venue2))]["VENUE-(A)-VENUE"] =
venue_edge_weight
                elif tuple((venue2,venue1)) in edge_weight_dict:
                    if "VENUE-(A)-VENUE" in edge_weight_dict[tuple((venue2,venue1))]:
                        edge_weight_dict[tuple((venue2,venue1))]["VENUE-(A)-VENUE"] +=
venue_edge_weight
                    else:
```

```
                        edge_weight_dict[tuple((venue2,venue1))]["VENUE-(A)-VENUE"] =
venue_edge_weight
                else:
                    print("None (else) condition entered for VENUE-A_VENUE edge!")
edge_set_topic_A_topic = set()
#TOPIC <-> (AUTHOR) <-> TOPIC Relation:  CO-PUB TOPIC:
for topic1 in topic_author_dict:
    for topic2 in topic_author_dict:
        if (topic1 != "NA") and (topic2 != "NA") and (topic1 != topic2) and (not
tuple((topic1,topic2)) in edge_set_topic_A_topic) and (not tuple((topic2,topic1)) in
edge_set_topic_A_topic):
            author_intersection =
set(topic_author_dict[topic1].keys()).intersection(set(topic_author_dict[topic2].keys()))
            topic_edge_weight = len(author_intersection)
            if topic_edge_weight != 0:
                if (not tuple((topic1,topic2)) in edge_set_topic_A_topic) and (not
tuple((topic2,topic1)) in edge_set_topic_A_topic): #As graph is multi-graph and undirected
so only need to represent edge between two - this would be a duplicate edge otherwise.
                    edge_set_topic_A_topic.add(tuple((topic1,topic2)))
                if not (tuple((topic1,topic2)) in edge_weight_dict) and (not
tuple((topic2,topic1)) in edge_weight_dict):
                    edge_weight_dict[tuple((topic1,topic2))] = {"TOPIC-(A)-TOPIC":
topic_edge_weight}
                elif tuple((topic1,topic2)) in edge_weight_dict:
                    if "TOPIC-(A)-TOPIC" in edge_weight_dict[tuple((topic1,topic2))]:
                        edge_weight_dict[tuple((topic1,topic2))]["TOPIC-(A)-TOPIC"] +=
topic_edge_weight
                    else:
                        edge_weight_dict[tuple((topic1,topic2))]["TOPIC-(A)-TOPIC"] =
topic_edge_weight
                elif tuple((topic2,topic1)) in edge_weight_dict:
                    if "TOPIC-(A)-TOPIC" in edge_weight_dict[tuple((topic2,topic1))]:
                        edge_weight_dict[tuple((topic2,topic1))]["TOPIC-(A)-TOPIC"] +=
topic_edge_weight
                    else:
                        edge_weight_dict[tuple((topic2,topic1))]["TOPIC-(A)-TOPIC"] =
topic_edge_weight
                else:
                    print("None (else) condition entered for TOPIC-(A)-TOPIC edge!")

#VENUE <-> TOPIC <-> VENUE (co-topic)
venue_topic_dict = {}
venue_set = set(Author_dataset["jconf"])
cnt = 1
for venue in venue_set:
    if venue != "NA" and venue != " ":
        venue_topic_dict[venue] = dict()
        print("Beginning {} of {} unique venues in building venue-topic
dictionary:".format(cnt,len(venue_set)))
        venue_topics = set()
        for index in range(len(Author_dataset)):
```

```
            topic_list = Author_dataset.iloc[index]["TOPICS"]
            author_venue = Author_dataset.iloc[index]["jconf"]
            if venue == author_venue:
                #add topics if proportion for topic is higher than 20%:
                for topic, prop in topic_list:
                    if prop > 0.20:
                        if topic in venue_topic_dict[venue]:
                            venue_topic_dict[venue][topic] += round(prop,2)
                        elif not topic in venue_topic_dict[venue]:
                            venue_topic_dict[venue][topic] = round(prop,2)
                        else:
                            print("FAILED DICT INCREMENT OF TOPIC",topic,"FOR VENUE",venue)
    cnt += 1
edge_set_venue_T_venue = set()
for venue1 in venue_topic_dict:
    for venue2 in venue_topic_dict:
        if (venue1 != "NA") and (venue2 != "NA") and (venue1 != venue2) and (not
tuple((venue1,venue2)) in edge_set_venue_T_venue):
            topic_intersection =
set(venue_topic_dict[venue1].keys()).intersection(set(venue_topic_dict[venue2].keys()))
            total_prop_intersection_weight = 0
            for topic in topic_intersection:
               venue1_top_prop = venue_topic_dict[venue1][topic]
               venue2_top_prop = venue_topic_dict[venue2][topic]
               total_prop_intersection_weight += min([venue1_top_prop,venue2_top_prop])
            if total_prop_intersection_weight != 0:
                if (not tuple((venue1,venue2)) in edge_set_venue_T_venue) and (not
tuple((venue2,venue1)) in edge_set_venue_T_venue): #As graph is multi-graph and undirected
so only need to represent edge between two - this would be a duplicate edge otherwise.
                    edge_set_venue_T_venue.add(tuple((venue1,venue2)))
                if not (tuple((venue1,venue2)) in edge_weight_dict) and (not
tuple((venue2,venue1)) in edge_weight_dict):
                    edge_weight_dict[tuple((venue1,venue2))] = {"VENUE-(T)-VENUE":
total_prop_intersection_weight}
                elif tuple((venue1,venue2)) in edge_weight_dict:
                    if "VENUE-(T)-VENUE" in edge_weight_dict[tuple((venue1,venue2))]:
                        edge_weight_dict[tuple((venue1,venue2))]["VENUE-(T)-VENUE"] +=
total_prop_intersection_weight
                    else:
                        edge_weight_dict[tuple((venue1,venue2))]["VENUE-(T)-VENUE"] =
total_prop_intersection_weight
                elif tuple((venue2,venue1)) in edge_weight_dict:
                    if "VENUE-(T)-VENUE" in edge_weight_dict[tuple((venue2,venue1))]:
                        edge_weight_dict[tuple((venue2,venue1))]["VENUE-(T)-VENUE"] +=
total_prop_intersection_weight
                    else:
                        edge_weight_dict[tuple((venue2,venue1))]["VENUE-(T)-VENUE"] =
total_prop_intersection_weight
                else:
                   print("None (else) condition entered for VENUE-T-VENUE edge!")
```

```
edge_type = "VENUE-(T)-VENUE"
for row in [[source,target,edge_type,edge_weight_dict[tuple((source,target))][edge_type]]
for source, target in edge_set_venue_T_venue if tuple((source, target)) in edge_weight_dict
and edge_type in edge_weight_dict[tuple((source,target))]]:
    print(row)
    nodes_edge_list_of_lists.append(row)
edge_type = "TOPIC-(A)-TOPIC"
for row in [[source,target,edge_type,edge_weight_dict[tuple((source,target))][edge_type]]
for source, target in edge_set_topic_A_topic if tuple((source, target)) in edge_weight_dict
and edge_type in edge_weight_dict[tuple((source,target))]]:
    print(row)
    nodes_edge_list_of_lists.append(row)

edge_type = "VENUE-(A)-VENUE"
for row in [[source,target,edge_type,edge_weight_dict[tuple((source,target))][edge_type]]
for source, target in edge_set_venue_A_venue if tuple((source, target)) in edge_weight_dict
and edge_type in edge_weight_dict[tuple((source,target))]]:
    print(row)
    nodes_edge_list_of_lists.append(row)


edge_type = "ORG-(A)-ORG"
for row in [[source,target,edge_type,edge_weight_dict[tuple((source,target))][edge_type]]
for source, target in edge_set_org_A_org if tuple((source, target)) in edge_weight_dict and
edge_type in edge_weight_dict[tuple((source,target))]]:
    print(row)
    nodes_edge_list_of_lists.append(row)

edge_type = "ORG-(P)-VENUE"
for row in [[source,target,edge_type,edge_weight_dict[tuple((source,target))][edge_type]]
for source, target in edge_set_org_P_venue if tuple((source, target)) in edge_weight_dict
and edge_type in edge_weight_dict[tuple((source,target))]]:
    print(row)
    nodes_edge_list_of_lists.append(row)


edge_type = "TOPIC-(P)-VENUE"
for row in [[source,target,edge_type,edge_weight_dict[tuple((source,target))][edge_type]]
for source, target in edge_set_topic_P_venue if tuple((source, target)) in edge_weight_dict
and edge_type in edge_weight_dict[tuple((source,target))]]:
    print(row)
    nodes_edge_list_of_lists.append(row)

edge_type = "ORG-(P)-TOPIC"
for row in [[source,target,edge_type,edge_weight_dict[tuple((source,target))][edge_type]]
for source, target in edge_set_org_P_topic if tuple((source, target)) in edge_weight_dict
and edge_type in edge_weight_dict[tuple((source,target))]]:
    print(row)
    nodes_edge_list_of_lists.append(row)

edge_type = "AUTHOR-(P)-AUTHOR"
```

```
for row in [[source,target,edge_type,edge_weight_dict[tuple((source,target))][edge_type]]
for source, target in edge_set_author_P_author if tuple((source, target)) in
edge_weight_dict and edge_type in edge_weight_dict[tuple((source,target))]]:
    print(row)
    nodes_edge_list_of_lists.append(row)

edge_type = "AUTHOR-(P)-VENUE"
for row in [[source,target,edge_type,edge_weight_dict[tuple((source,target))][edge_type]]
for source, target in edge_set_author_P_venue if tuple((source, target)) in
edge_weight_dict and edge_type in edge_weight_dict[tuple((source,target))]]:
    print(row)
    nodes_edge_list_of_lists.append(row)

edge_type = "AUTHOR-(P)-ORG"
for row in [[source,target,edge_type,edge_weight_dict[tuple((source,target))][edge_type]]
for source, target in edge_set_author_P_ORG if tuple((source, target)) in edge_weight_dict
and edge_type in edge_weight_dict[tuple((source,target))]]:
    print(row)
    nodes_edge_list_of_lists.append(row)


edge_type = "AUTHOR-(P)-TOPIC"
for row in [[source,target,edge_type,edge_weight_dict[tuple((source,target))][edge_type]]
for source, target in edge_set_author_P_topic if tuple((source, target)) in
edge_weight_dict and edge_type in edge_weight_dict[tuple((source,target))]]:
    print(row)
    nodes_edge_list_of_lists.append(row)

##edge_type = "AUTHOR-(T)-AUTHOR"
#for row in [[source,target,edge_type,edge_weight_dict[tuple((source,target))][edge_type]]
for source, target in edge_set_author_T_author if tuple((source, target)) in
edge_weight_dict and edge_type in edge_weight_dict[tuple((source,target))]]:
#    print(row)
#    nodes_edge_list_of_lists.append(row)


#Converting and pickling dataset into a dataframe containing each relation, edge set and
weight instance between the networks.
nodes_edges_df = pd.DataFrame(nodes_edge_list_of_lists, columns = ["source", "target",
"relation","weight"])
set(nodes_edges_df.relation)
pickling_on = open("Hetergenous_dataset_final.pickle","wb")
pickle.dump(nodes_edges_df, pickling_on)
pickling_on.close()
```

Script 5.2 Implementation of Metapath embedding, Link Prediction and Research Collaboration Recommendations

```
"""
Link Prediction (Conversion of dataframe to Stellargraph and basic link prediction using
                 Metapath2Vec & Logistic Regression Classifier) based on Stellargraph
link prediction tutorial/documentation
                 of static graph data - not utilising year temporal data so far - in
future this will be extended
                 to a dynamic graph training on 4 years and testing on the subsequent 2
years,
                 with the most recent calendar year being the development set for real-
time recommendations.
Tutorial from StellarGraph Metapath2vec & Link Prediction heavily informed method –
https://stellargraph.readthedocs.io/en/stable/demos/link-prediction/metapath2vec-link-
prediction.html
Kaggle Tutorial for LightGBM Hyperparameter training, validation & testing heavily informed
training/testing of LightGBM – https://www.kaggle.com/mlisovyi/lightgbm-hyperparameter-
optimisation-lb-0-761
"""
```

```
#Importing Libraries:
import sys
import pickle
import pandas as pd
import matplotlib.pyplot as plt
from math import isclose
from sklearn.decomposition import PCA
import os
import networkx as nx
import numpy as np
import pandas as pd
from stellargraph import StellarGraph, datasets
from stellargraph.data import EdgeSplitter
from collections import Counter
import multiprocessing
from IPython.display import display, HTML
from sklearn.model_selection import train_test_split

#Getting primary author names:
primary_authors = pd.read_csv("C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/2-3 - Disambiguation and Network Comparison
(IGM)/Disambiguation_summary.csv")
primary_authors_set = set(primary_authors.Name)

#Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/5-Link
Prediction(Recommendations)/1-Preparing multitype nodes and edge
dataset/Hetergenous_dataset_final.pickle",'rb')
unpickler = pickle.Unpickler(file)
```

```
Hetero_dataset = unpickler.load()


#Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic
Representation/1-Secondary Author Pruning & Topic Modelling/Final_dataset.pickle",'rb')
#Final_dataset = pickle.load(file)
unpickler = pickle.Unpickler(file)
Author_dataset = unpickler.load()


#Creating different node types:
org_nodes = pd.DataFrame(index=[org for org in set(Author_dataset["organization"]) if org
!= "NA" and org!='palliative medicine'])
#bug in extracted dataset - org!='palliative medicine' - duplicated venue in org - needs
manually cleaning to be filtered out for stellargraph dataset creation.
author_nodes = [author for author in set(Author_dataset["name"]) if author != "NA"]
author_nodes = set(author_nodes)
for coauthors in set(Author_dataset["coauthors"]):
    coauthors = coauthors.split(" ")
    for coauthor in coauthors:
        if not coauthor in author_nodes:
            author_nodes.add(coauthor)
author_nodes = pd.DataFrame(index=author_nodes)
venue_nodes = pd.DataFrame(index=[jconf for jconf in set(Author_dataset["jconf"]) if jconf
!= "NA"])
topic_csv = pd.read_csv("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic
Representation/1-Secondary Author Pruning & Topic
Modelling/final_dataset_TPC_Modelled_Summary.csv")
topic_set = set()
for topic_list in Author_dataset["TOPICS"]:
    for topic, prop in topic_list:
        topic_set.add("T{}".format(str(int(topic) + 1)))
topic_nodes =  pd.DataFrame(index=topic_set)
#adding in missing topic nodes - need to fix this bug at some point!- some topics are being
pruned out.
import re
try:
    #Creating the multi-node and edge Stellargraph (hetergenous graph):
    author_hetero_weighted = StellarGraph(
    {"author": author_nodes,"venue": venue_nodes, "org": org_nodes, "topic":topic_nodes},
    Hetero_dataset,
    edge_type_column="relation",
)
except:
    the_type, the_value, the_traceback = sys.exc_info()
    if str(the_type) == "<class 'ValueError'>" and str(the_value).split(":")[1] ==  '
expected all source and target node IDs to be contained in `nodes`, found some missing':
        the_value = str(the_value)
        missing_topics = re.findall("T\d+",the_value)
        for topic in missing_topics:
```

```
            topic_set.add(topic)
            topic_nodes =  pd.DataFrame(index=topic_set)
            #Creating the multi-node and edge Stellargraph (hetergenous graph):
        author_hetero_weighted = StellarGraph(
                    {"author": author_nodes,"venue": venue_nodes, "org": org_nodes,
"topic":topic_nodes},
                    Hetero_dataset,
                    edge_type_column="relation",
                    )
print(author_hetero_weighted.info())
#We have to carefully split the data to avoid data leakage and evaluate the algorithms
correctly:
#For computing node embeddings, a Train Graph (graph_train)

#For training classifiers, a classifier Training Set (examples_train) of positive and
negative edges that were not used for computing node embeddings
#For choosing the best classifier, a Model Selection Test Set (examples_model_selection) of
positive and negative edges that were not used for computing node embeddings or training
the classifier
#For the final evaluation, a Test Graph (graph_test) to compute test node embeddings with
more edges than the Train Graph, and a Test Set (examples_test) of positive and negative
edges not used for neither computing the test node embeddings or for classifier training or
model selection
#We begin with the full graph and use the EdgeSplitter class to produce Test Graph & Train
Graph

#Splitting edges for testing data or the testing graph: --- TEST GRAPH EDGES
#Test Graph - Test set of positive/negative link examples
#The Test Graph is the reduced graph we obtain from removing the test set of links from the
full graph.
# Define an edge splitter on the original graph:
edge_splitter_test = EdgeSplitter(author_hetero_weighted)
# Randomly sample a fraction p=0.1 of all positive links, and same number of negative
links, from graph, and obtain the
# reduced graph graph_test with the sampled paper links (indicating collaborations)
removed:
    #Test Graph
    #examples_test: Test set of positive/negative link examples
graph_test, examples_test, labels_test = edge_splitter_test.train_test_split(
    p=0.08, method="global", edge_label="AUTHOR-(P)-AUTHOR"
)
NOTE: COULD NOT USE HIGHER THAN 0.8% SAMPLING DUE TO NOT ENOUGH POS/NEG EDGES
print(graph_test.info())
# TRAINING GRAPH
#This time, we use the EdgeSplitter on the Test Graph, and perform a train/test split on
the examples to produce, like test graph, producing:
  #1. Training graph (train graph)
  #2. examples: Training set of link examples
  #3. labels: Set of link examples for model selection

#Splitting edges for testing data or the testing graph:
```

```
edge_splitter_train = EdgeSplitter(graph_test, author_hetero_weighted)
graph_train, examples, labels = edge_splitter_train.train_test_split(
    p=0.08, method="global", edge_label="AUTHOR-(P)-AUTHOR"
)
(
    examples_train,
    examples_model_selection,
    labels_train,
    labels_model_selection,
) = train_test_split(examples, labels, train_size=0.75, test_size=0.25)
print(graph_train.info())

# Do the same process to compute a training subset from within the test graph
edge_splitter_train = EdgeSplitter(graph_test, author_hetero_weighted)
graph_train, examples, labels = edge_splitter_train.train_test_split(
    p=0.08, method="global", edge_label="AUTHOR-(P)-AUTHOR"
)
(
    examples_train,
    examples_model_selection,
    labels_train,
    labels_model_selection,
) = train_test_split(examples, labels, train_size=0.75, test_size=0.25)
print(graph_train.info())

#Defining parameters for metapath2vec embeddings & defining metapaths:
dimensions = 150
num_walks = 5
walk_length = 120
context_window_size = 15
num_iter = 3
workers = multiprocessing.cpu_count()

#DEFINING META-PATHS FOR METAPATH2VEC RANDOM-WALK EMBEDDINGS:
user_metapaths = [
    #Adding in metapaths for each node-type pathway:
    ["author","org","author"], #co-organisation
    ["author","author"], #co-authorship
    ["author","topic","author"], #co-topic
    ["author","venue","author"], #co-venue
    ["author","org","venue","topic","venue","org","author"], #global topic meta-path
    ["org","author","author","org"],  #  organisation/author co-author interactions
    ["venue","author","author","venue"] #  organisation/author topic interactions
]
#Training the graph:
from stellargraph.data import UniformRandomMetaPathWalk
from gensim.models import Word2Vec
def metapath2vec_embedding(graph, name):
    rw = UniformRandomMetaPathWalk(graph)
    walks = rw.run(
        graph.nodes(), n=num_walks, length=walk_length, metapaths=user_metapaths
```

```
    )
    print(f"Number of random walks for '{name}': {len(walks)}")

    model = Word2Vec(
        walks,
        size=dimensions,
        window=context_window_size,
        min_count=0,
        sg=1,
        workers=workers,
        iter=num_iter,
    )

    def get_embedding(u):
        return model.wv[u]

    return get_embedding



embedding_train = metapath2vec_embedding(graph_train, "Train Graph")

from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import lightgbm as lgbm
from sklearn.preprocessing import StandardScaler


# 1. link embeddings
def link_examples_to_features(link_examples, transform_node, binary_operator):
    return [
        binary_operator(transform_node(src), transform_node(dst))
        for src, dst in link_examples
    ]
# 2. training classifier
def train_link_prediction_model(
    link_examples, link_labels, get_embedding, binary_operator
):
    clf = link_prediction_classifier()
    link_features = link_examples_to_features(
        link_examples, get_embedding, binary_operator
    )
    clf.fit(link_features, link_labels)
    return clf
def link_prediction_classifier(max_iter=2000):
    lr_clf = LogisticRegressionCV(Cs=10, cv=10, scoring="roc_auc", max_iter=max_iter)
    return Pipeline(steps=[("sc", StandardScaler()), ("clf", lr_clf)])

# 3. and 4. evaluate classifier
```

```
def evaluate_link_prediction_model(
    clf, link_examples_test, link_labels_test, get_embedding, binary_operator
):
    link_features_test = link_examples_to_features(
        link_examples_test, get_embedding, binary_operator
    )
    score = evaluate_roc_auc(clf, link_features_test, link_labels_test)
    lr_fpr, lr_tpr, _ = create_roc_curve(clf, link_features_test, link_labels_test)
    return score,lr_fpr, lr_tpr


def evaluate_roc_auc(clf, link_features, link_labels):
    predicted = clf.predict_proba(link_features)

    # check which class corresponds to positive links
    positive_column = list(clf.classes_).index(1)
    return roc_auc_score(link_labels, predicted[:, positive_column])

def create_roc_curve(clf, link_features, link_labels):
    predicted = clf.predict_proba(link_features)
    # check which class corresponds to positive links
    positive_column = list(clf.classes_).index(1)
    return roc_curve(link_labels, predicted[:, positive_column])

def operator_L1(u, v):
    return np.abs(u - v)

def operator_L2(u, v):
    return (u - v) ** 2

def run_link_prediction(binary_operator):
    clf = train_link_prediction_model(
        examples_train, labels_train, embedding_train, binary_operator
    )
    score,lr_fpr, lr_tpr = evaluate_link_prediction_model(
        clf,
        examples_model_selection,
        labels_model_selection,
        embedding_train,
        binary_operator,
    )

    return {
        "lr_fpr":lr_fpr,
        "lr_tpr":lr_tpr,
        "classifier": clf,
        "binary_operator": binary_operator,
        "score": score,
    }

binary_operators = [operator_L1, operator_L2]
```

```
results = [run_link_prediction(op) for op in binary_operators]
best_result = max(results, key=lambda result: result["score"])

print(f"Best result from '{best_result['binary_operator'].__name__}'")

pd.DataFrame(
    [(result["binary_operator"].__name__, result["score"]) for result in results],
    columns=("name", "ROC AUC score"),
).set_index("name")

embedding_test = metapath2vec_embedding(graph_test, "Test Graph")

#Formatting for training and testing of a Gradient-boosted light model:
# Feature Scaling for light gradient boosted models:
sc = StandardScaler()
train_features_best = np.array(link_examples_to_features(examples_train, embedding_train,
best_result['binary_operator']))
train_features_best = sc.fit_transform(train_features_best)
test_features_best = np.array(link_examples_to_features(examples_test, embedding_test,
best_result['binary_operator']))
test_features_best = sc.fit_transform(test_features_best)


def learning_rate_010_decay_power_0995(current_iter):
    base_learning_rate = 0.1
    lr = base_learning_rate  * np.power(.995, current_iter)
    return lr if lr > 1e-3 else 1e-3

import lightgbm as lgb
fit_params={"early_stopping_rounds":30,
            "eval_metric" : 'auc',
            "eval_set" : [(test_features_best,labels_test)],
            'eval_names': ['valid'],
            #'callbacks':
[lgb.reset_parameter(learning_rate=learning_rate_010_decay_power_099)],
            'verbose': 100,
            'categorical_feature': 'auto'}

from scipy.stats import randint as sp_randint
from scipy.stats import uniform as sp_uniform
param_test ={'num_leaves': sp_randint(6, 50),
             'min_child_samples': sp_randint(100, 500),
             'min_child_weight': [1e-5, 1e-3, 1e-2, 1e-1, 1, 1e1, 1e2, 1e3, 1e4],
             'subsample': sp_uniform(loc=0.2, scale=0.8),
             'colsample_bytree': sp_uniform(loc=0.4, scale=0.6),
             'reg_alpha': [0, 1e-1, 1, 2, 5, 7, 10, 50, 100],
             'reg_lambda': [0, 1e-1, 1, 5, 10, 20, 50, 100]}

#This parameter defines the number of HP points to be tested
n_HP_points_to_test = 100
```

```
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
#n_estimators is set to a "large value". The actual number of trees build will depend on
early stopping and 5000 define only the absolute maximum
clf = lgb.LGBMClassifier(max_depth=-1, random_state=314, silent=True, metric='None',
n_jobs=4, n_estimators=5000)
gs = RandomizedSearchCV(
    estimator=clf, param_distributions=param_test,
    n_iter=n_HP_points_to_test,
    scoring='roc_auc',
    cv=3,
    refit=True,
    random_state=314,
    verbose=True)
gs.fit(train_features_best, labels_train, **fit_params)
print('Best score reached: {} with params: {} '.format(gs.best_score_, gs.best_params_))
clf_sw = lgb.LGBMClassifier(**clf.get_params())
#set optimal parameters
clf_sw.set_params(**gs.best_params_)
gs_sample_weight = GridSearchCV(estimator=clf_sw,
                                param_grid={'scale_pos_weight':[1,2,6,12]},
                                scoring='roc_auc',
                                cv=5,
                                refit=True,
                                verbose=True)

gs_sample_weight.fit(train_features_best, labels_train, **fit_params)
print('Best score reached: {} with params: {} '.format(gs_sample_weight.best_score_,
gs_sample_weight.best_params_))

#Configure from the HP optimisation
clf_final = lgb.LGBMClassifier(**gs.best_estimator_.get_params())
#Train the final model with learning rate decay
clf_final.fit(train_features_best, labels_train, **fit_params,
callbacks=[lgb.reset_parameter(learning_rate=learning_rate_010_decay_power_0995)])

#Predictions for light gradient boosted model for embeddings as L1 and L2 regularization:
y_pred_hyp=clf_final.predict_proba(test_features_best)
# check which class corresponds to positive links
positive_column = list(clf_final.classes_).index(1)
y_pred_hyp = y_pred_hyp[:, positive_column]

gbhyp_fpr, gbhyp_tpr, _ = roc_curve(labels_test, y_pred_hyp)
gbhyp_auc = roc_auc_score(labels_test, y_pred_hyp)


# generating a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(labels_test))]
ns_fpr, ns_tpr, _ = roc_curve(labels_test, ns_probs)
ns_auc = roc_auc_score(labels_test, ns_probs)
lr_name = "Logistic Regression -" + " " +
best_result['binary_operator'].__name__.split("_")[1] + " regularization"
```

```
gb_name = "Light Gradient Boosting Machine(LightGBM) -" + " " +
best_result['binary_operator'].__name__.split("_")[1] + " regularization"
auc_curves = {"No Skill":[ns_fpr, ns_tpr],
              gb_name:[gbhyp_fpr, gbhyp_tpr]
             }

#Evaluating both models on test dataset for plotting RUC Curve:
test_scores = dict()
print("Beginning evaluation on test dataset for", best_result['binary_operator'].__name__)
score,lr_fpr, lr_tpr = evaluate_link_prediction_model(best_result["classifier"],
                                                      examples_test,
                                                      labels_test,
                                                      embedding_test,
                                                     best_result['binary_operator'])
auc_curves[lr_name] = [lr_fpr, lr_tpr]
test_scores[lr_name] = score

#Adding score of no-skill model and gradient boosted model:
test_scores["No Skill"] = ns_auc
test_scores[gb_name] = gbhyp_auc

for classifier in test_scores:
    print("{}: ROC AUC={}".format(classifier,test_scores[classifier]))




from matplotlib import pyplot
#Plotting ROC Curves for both logistic regression models and no-skill baseline:
pyplot.plot(auc_curves['No Skill'][0], auc_curves['No Skill'][1],
            linestyle='--', label= 'No Skill')
pyplot.plot(auc_curves[lr_name][0], auc_curves[lr_name][1], marker='.', label=lr_name)
pyplot.plot(auc_curves[gb_name][0], auc_curves[gb_name][1], marker='.', label=gb_name)

# axis labels
pyplot.xlabel('False Positive Rate (1 - Specificity)')
pyplot.ylabel('True Positive Rate (Sensitivity)')
# show & save the plot
pyplot.legend()
pyplot.savefig("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/Evaluation
Results and Tables/E2-Link Prediction(Recommendations) Results/OUTPUT-
ROC_Curve_Comparison",
               dpi=None, facecolor='w', edgecolor='w',
        orientation='portrait', papertype=None, format=None,
        transparent=False, bbox_inches=None, pad_inches=0.1,
        frameon=None, metadata=None)
pyplot.show()

#Pulling out predicted recommendations dataframe using best model:
max_score = 0
best_classifier = ''
```

```
for classifier in test_scores:
    if test_scores[classifier] > max_score:
        max_score = test_scores[classifier]
        best_classifier = classifier
if best_classifier == lr_name:

    # Calculate edge features for test data using best model:
    link_features = link_examples_to_features(
    examples_test, embedding_test, best_result["binary_operator"])
    predicted_link_labels = best_result["classifier"].predict(link_features)
    predicted_link_prob = best_result["classifier"].predict_proba(link_features)
    positive_column = list(best_result["classifier"].classes_).index(1)
    print("Using logistic regression model for link prediction/recommendations")
else:
    if best_classifier == gb_name:
         # Calculate edge features for test data using best model:
        link_features = link_examples_to_features(
        examples_test, embedding_test, best_result["binary_operator"])
        predicted_link_labels = clf_final.predict(link_features)
        predicted_link_prob = clf_final.predict_proba(link_features)
        positive_column = list(clf_final.classes_).index(1)
        print("Using Light Gradient Boosting Machine (LightGBM) model for link
prediction/recommendations")
#labels_test: Labels for each node-pair
predicted_link_prob = predicted_link_prob[:, positive_column]
#examples_test: author-pair names (examples)
def recommendations_to_df(predicted_labels, pos_predicted_probabilities, true_label,
author_pair_names):
    recommendation_list_of_lists = []
    if (len(predicted_labels) == len(author_pair_names) and
len(pos_predicted_probabilities) == len(author_pair_names)
                              and len(true_label) == len(author_pair_names) ):
        for index in range(len(author_pair_names)):
            if (author_pair_names[index][0] in primary_authors_set or
author_pair_names[index][1] in primary_authors_set):
                recommendation_list = []
                recommendation_list.append(predicted_labels[index])
                recommendation_list.append(pos_predicted_probabilities[index])
                recommendation_list.append(true_label[index])
                recommendation_list.append(author_pair_names[index][0])
                recommendation_list.append(author_pair_names[index][1])
                recommendation_list_of_lists.append(recommendation_list)
        return recommendation_list_of_lists
    else:
        raise NameError('Vectors are not of equal length!')
recommendations_list = recommendations_to_df(predicted_link_labels, predicted_link_prob,
labels_test,examples_test)
recommendations_df = pd.DataFrame(recommendations_list, columns = ["Predicted_Label",
"Pos_Pred_Prob", "True_Link_label","Coauthor1", "Coauthor2"])
#Saving Primary Organisation Recommendations dataset as pickle for later usage:
pickling_on = open("Hetergenous_Recommendations.pickle","wb")
```

```
pickle.dump(recommendations_df, pickling_on)
pickling_on.close()
#Creating PCA Visualisation as diagnostic output:
# Learn a projection from 128 dimensions to 2
pca = PCA(n_components=2)
X_transformed = pca.fit_transform(link_features)

# plot the 2-dimensional points
plt.figure(figsize=(16, 12))
plt.scatter(
    X_transformed[:, 0],
    X_transformed[:, 1],
    c=np.where(labels_test == 1, "b", "r"),
    alpha=0.5,
)
#Saving output of PCA for embedded nodes and edges (links)
plt.savefig("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/Evaluation Results
and Tables/E2-Link Prediction(Recommendations) Results/OUTPUT-PCA_NODES_EDGES_LINK_PRED",
dpi=None, facecolor='w', edgecolor='w',
        orientation='portrait', papertype=None, format=None,
        transparent=False, bbox_inches=None, pad_inches=0.1,
        frameon=None, metadata=None)
```

Script 5.3 Preparing In-direct Co-authorship Recommendation Inference Visualization files for MuxViz

Preferably this would be undertaken in Rshiny interactively, with user selecting a Research Recommendation on Rshiny front-end and with back-end running this script file to produce a MuxViz file for specific research collaboration and then calling MuxViz with set parameters to then Visualize the co-authorship network for the recommendation researcher pair.

```
#Import Libraries:
import pickle
import pandas as pd
import matplotlib.pyplot as plt
from math import isclose
import os

#Getting primary author names:
primary_authors = pd.read_csv("C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/2-3 - Disambiguation and Network Comparison
(IGM)/Disambiguation_summary.csv")
primary_authors = set(primary_authors.Name)

#Getting two recommendation author names:
pair_list = []
with open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/5-Link
Prediction(Recommendations)/Indirect-Network-Recommendation-Vizualisations/Indirect-
relationship-recommendation-pairs.txt", "r") as file:
    for line in file.readlines():
        print(line)
        pair_list.append(line[:-1])
for pair in pair_list:
    primary_authors_set = set()
    author1,author2 = pair.split("|")
    if author1 in primary_authors:
        primary_authors_set.add(author1)
    if author2 in primary_authors:
        primary_authors_set.add(author2)

    #Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
    file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/5-Link
Prediction(Recommendations)/1-Preparing multitype nodes and edge
dataset/Hetergenous_dataset_final.pickle",'rb')
    unpickler = pickle.Unpickler(file)
    Hetero_dataset = unpickler.load()
    Hetero_dataset = Hetero_dataset.loc[Hetero_dataset.weight > 1]
    Hetero_dataset = Hetero_dataset[Hetero_dataset.relation == 'AUTHOR-(P)-AUTHOR']

    #Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
    file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic
Representation/1-Secondary Author Pruning & Topic Modelling/Final_dataset.pickle",'rb')
    unpickler = pickle.Unpickler(file)
    Author_dataset = unpickler.load()
    #Creating different node types:
```

```
    author_nodes = [author.replace(" ", "_") for author in set(Author_dataset["name"]) ]
    author_nodes = set(author_nodes)
    for coauthors in set(Author_dataset["coauthors"]):
        coauthors = coauthors.split(" ")
        for coauthor in coauthors:
            if not coauthor in author_nodes:
                author_nodes.add(coauthor)
    author_nodes = pd.DataFrame(index=author_nodes)
    import os
    directory = "C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/6-Network
Vizualisation/2-Heterogeneous-Multiplex Vizualizations MuxVix File Creation"
    os.chdir(directory + "/muxViz_network_files")
    relations = {"Authors"}
    layer_dict = dict()
    cnt = 0
    for relation in relations:
        cnt += 1
        layer_dict[relation] = cnt

    with open("./muxViz-master/MultiGraph_Layers_{}-{}.txt".format(author1,author2), "w",
encoding="utf-8") as file:
        file.write("layerID layerLabel\n")
        for layer in layer_dict.keys():
            row_str = "{} {}\n".format(layer_dict[layer], layer)
            print(row_str)
            file.write(row_str)
        file.close()

    #Loading in Link Prediction results for recommendations:
    file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/5-Link
Prediction(Recommendations)/2-Implementation of Metapath embedding and Link
Prediction/Hetergenous_Recommendations.pickle",'rb')
    unpickler = pickle.Unpickler(file)
    Recommendations_dts = unpickler.load()

    #Removing recommendations where label predicted is negative (not recommended to
collaborate):
    Recommendations_dts = Recommendations_dts[Recommendations_dts.Predicted_Label == 1]
    #Looking for any matching co-author pair who have been recommended to collaborate in
the development network dataset and do not yet share a co-author edge (i.e. have no co-
authored a paper together and therefore will not be in hetero-edge dataset.

    #Building coauthor set for hetergenous dataset to check against for new collaboration
pairs:
    coauthor_set = set()
    for source, target, relation, weight in Hetero_dataset[Hetero_dataset.relation ==
'AUTHOR-(P)-AUTHOR'].values:
        if tuple((source, target)) not in coauthor_set:
            coauthor_set.add(tuple((source, target)))
    print(len(coauthor_set), "unique coauthor pairs")
```

```
    #note: will remove true label in final version as there will be no true label for
current pub year only predicted labels.
    raw_cnt = 0
    prob_filtered_cnt = 0
    primary_recommendations = 0
    secondary_recommendations = 0

    primary_secondary_recommendations_list = []
    for pred_label, pred_prob, true_label, author_1, author_2 in
Recommendations_dts.values:
        if ((author_1 == author1) and (author_2 == author2)) or ((author_2 == author1) and
(author_1 == author2)):
            if (not tuple((author_1,author_2)) in coauthor_set) and (not
tuple((author_2,author_1)) in coauthor_set):
                raw_cnt += 1
                #print(pred_prob)
                if pred_prob > 0.00:
                    prob_filtered_cnt += 1
                    if ((author_1 in primary_authors_set) or (author_2 in
primary_authors_set)) and not ((author_1 in primary_authors_set) and (author_2 in
primary_authors_set)):
                        secondary_recommendations += 1
                        primary_secondary_recommendations_list.append([pred_prob, author_1,
author_2, "secondary"])
                    elif (author_1 in primary_authors_set) and (author_2 in
primary_authors_set):
                            primary_recommendations += 1
                            primary_secondary_recommendations_list.append([pred_prob,
author_1, author_2, "primary"])

    #Creating node density dictionaries to calculate standardised degree for each node-
default 0 (no edges):
    author_density_dict = dict()
    for author in author_nodes.index:
        author_density_dict[author] = 0
    for source, target, relation, weight in Hetero_dataset.values:
        if source.replace(" ", "_") in author_density_dict:
            author_density_dict[source.replace(" ", "_")] += 1
        if target.replace(" ", "_") in author_density_dict:
            author_density_dict[target.replace(" ", "_")] += 1
    author_node_length = len(author_nodes.index)

    #Standardizing each node type degree, seperately by node type:
    for author in author_density_dict:
        author_density_dict[author] = author_density_dict[author]/(author_node_length-1)
    import numpy as np
        percentile = 99
    #Taking {percentile} as threshold cut-off value for network nodes and edges:
    author_density_threshold = np.percentile(list(author_density_dict.values()),
percentile)
    max_weight = max(list(Hetero_dataset["weight"]))
```

```
    #Normalizing edge weights:
    Hetero_dataset["weight"] = Hetero_dataset["weight"]/max_weight
    average_edge_weight = np.mean(Hetero_dataset["weight"])
    edge_weight_threshold = np.percentile(list(Hetero_dataset["weight"]),percentile)
    max_weight_normalized = max(list(Hetero_dataset["weight"]))
    #Creating recommendation set for checking authors:
    auth_recommended_dict = dict()
    auth_recommended_dict["primary"] = set()
    auth_recommended_dict["secondary"] = set()
    auth_recommended_dict["primary_pairs"] = set()
    auth_recommended_dict["secondary_pairs"] = set()

    for prob, author_1, author_2, rec_type in primary_secondary_recommendations_list:
        if ((author_1 == author1) and (author_2 == author2)) or ((author_2 == author1) and
(author_1 == author2)):
            if rec_type == "primary":
                auth_recommended_dict["primary"].add(author1)
                auth_recommended_dict["primary"].add(author2)
                auth_recommended_dict["primary_pairs"].add(tuple((author1, author2)))

            elif rec_type == "secondary":
                auth_recommended_dict["secondary"].add(author1)
                auth_recommended_dict["secondary"].add(author2)
                auth_recommended_dict["secondary_pairs"].add(tuple((author1, author2)))

    Primary_author_dict = dict()
    Primary_edge_dict = dict()
    for source, target, relation, weight in Hetero_dataset.values:
            if ((source in primary_authors_set ) or (target in primary_authors_set)):
                #Checking if author is a primary researcher or not:
                if source in primary_authors_set and not source in Primary_author_dict:
                    Primary_author_dict[source] = True
                elif not source in primary_authors_set and not source in
Primary_author_dict:
                    Primary_author_dict[author] = False
                if target in primary_authors_set and not target in Primary_author_dict:
                    Primary_author_dict[target] = True
                elif not target in primary_authors_set and not target in
Primary_author_dict:
                    Primary_author_dict[target] = False
                #Checking if a primary edge or not:
                if not tuple((source, target)) in Primary_edge_dict and not tuple((target,
source)) in Primary_edge_dict:
                    Primary_edge_dict[tuple((source, target))] = (True,weight,False)
                else:
                    print("EXCEPTION-4",(tuple((source, target)) in Primary_edge_dict or
tuple((target, source)) in Primary_edge_dict ))
            elif ((not source in primary_authors_set) and (not target in
primary_authors_set)):
                Primary_author_dict[source] = False
                Primary_author_dict[target] = False
```

```
                if not tuple((source, target)) in Primary_edge_dict and not tuple((target,
source)) in Primary_edge_dict:
                    Primary_edge_dict[tuple((source, target))] = (False,weight, False)
            else:
                print("EXCEPTION5",source, target, relation, weight)
    for author_1,author_2 in auth_recommended_dict["primary_pairs"]:
        #Primary authors already put into dictionary, thereby can assume if not in Primary
dict than is a secondary author.
        if not author_1 in Primary_author_dict:
            Primary_author_dict[author_1] = False
        if not author_2 in Primary_author_dict:
            Primary_author_dict[author_2] = False
        if not tuple((author_1,author_2)) in Primary_edge_dict and not
tuple((author_2,author_1)) in Primary_edge_dict:
            Primary_edge_dict[tuple((author_1,author_2))] = (True, edge_weight_threshold,
"primary_pairs")

    for author_1,author_2 in auth_recommended_dict["secondary_pairs"]:
        #All authors in secondary pair are secondary authors.
        if not author_1 in Primary_author_dict:
            Primary_author_dict[author_1] = False
        if not author_2 in Primary_author_dict:
            Primary_author_dict[author_2] = False
        if not tuple((author_1,author_2)) in Primary_edge_dict and not
tuple((author_2,author_1)) in Primary_edge_dict:
            Primary_edge_dict[tuple((author_1,author_2))] = (True, edge_weight_threshold,
"secondary_pairs")
    index_count = 0
    index_dict = dict()
    with open("./muxViz-master/MultiGraph_node_layout_{}-{}.txt".format(author1,author2),
"w", encoding="utf-8") as file:
        file.write("nodeID nodeLabel\n")
        for author in Primary_author_dict:
                #if (Primary_author_dict[author] == True and author in
primary_authors_set):
                   #   or (author in auth_recommended_dict["primary"]) or  (author in
auth_recommended_dict["secondary"]):
                index_count += 1
                index_dict[author] = index_count
                row_str = "{} {}\n".format(index_dict[author], author)
                print(row_str)
                file.write(row_str)
        file.close()
    print(index_count)
    #Creating node colour and size file - NEW version only primary nodes and non-primary
nodes above threshold for degree:
    cnt = 0
    recom_author_cnt = 0
    with open("./muxViz-master/MultiGraph_node_colour_size_{}-
{}.txt".format(author1,author2), "w", encoding="utf-8") as file:
        file.write("nodeID layerID color size\n")
```

```
        for node in Primary_author_dict:
            if (node in primary_authors_set) and (not (node in
auth_recommended_dict["secondary"])) and (not (node in auth_recommended_dict["primary"])):
                        row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Authors"], "orange",author_density_dict[node]*5)
                       # print(row_str)
                        cnt += 1
                        file.write(row_str)
            elif (node in primary_authors_set) and ((node in
auth_recommended_dict["secondary"]) or ( node in auth_recommended_dict["primary"])):
                        print("Recommendation primary node:", node)
                        row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Authors"], "red",author_density_dict[node]*5)
                        #print(row_str)
                        cnt += 1
                        file.write(row_str)
            elif (not node in primary_authors_set) and ((node in
auth_recommended_dict["secondary"]) or (node in auth_recommended_dict["primary"])):
                        print("Recommendation secondary node:", node)
                        row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Authors"], "purple",(author_density_dict[node] + author_density_threshold)*5)
                        #print(row_str)
                        cnt += 1
                        file.write(row_str)
            else:
                        row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Authors"], "green",author_density_dict[node]*5)
                        #print(row_str)
                        cnt += 1
                        file.write(row_str)

        file.close()



    #Creating edge size and colour file:
    #FORMAT: nodeID.from layerID.from nodeID.to layerID.to color size
    cnt = 0
    with open("./muxViz-master/MultiGraph_edge_colour_size_{}-
{}.txt".format(author1,author2), "w", encoding="utf-8") as file:
        file.write("nodeID.from layerID.from nodeID.to layerID.to color size\n")
        for author_1, author_2 in Primary_edge_dict:
            primary_flag, weight, pair_bool = Primary_edge_dict[tuple((author_1,
author_2))]
            #Giving emphasis in primary author edges in Vizualisation making edges
different colours and of significantly higher weight:
            #Looking for primary author edge instances:
            if  primary_flag == True:
                if pair_bool == "primary_pairs":
                    #This is a primary recommendation pair edge between two primary authors
(Blue):
```

```
                    layerID_from = layer_dict["Authors"]
                    layerID_to = layer_dict["Authors"]
                    row_str = "{} {} {} {} {}
{}\n".format(index_dict[author_1],layerID_from,
                                                            index_dict[author_2],
layerID_to,
                                                            "blue", weight)
                    file.write(row_str)
                    cnt += 1
                    print("primary_pair edge")
                elif pair_bool == "secondary_pairs":
                    #This is a secondary recommendation pair edge with a primary author
(Purple):
                    layerID_from = layer_dict["Authors"]
                    layerID_to = layer_dict["Authors"]
                    row_str = "{} {} {} {} {}
{}\n".format(index_dict[author_1],layerID_from,
                                                            index_dict[author_2],
layerID_to,
                                                            "red", weight)
                    file.write(row_str)
                    cnt += 1
                    print("secondary_pair edge")
                else: #pair_bool == False NOT a recommendation edge, although IS a normal
primary edge:
                    #This is a normal primary edge (ORANGE):
                    layerID_from = layer_dict["Authors"]
                    layerID_to = layer_dict["Authors"]
                    row_str = "{} {} {} {} {}
{}\n".format(index_dict[author_1],layerID_from,
                                                            index_dict[author_2],
layerID_to,
                                                            "orange", weight)
                    file.write(row_str)
                    cnt += 1

            elif  primary_flag == False:
                cnt += 1 #Not a primary edge, thereby edge will take on default secondary
colour set in MuxViz.
            else:
                print("EXCEPTION: Edge Colour & Size")
        file.close()
    print("{} edges".format(cnt))


    #[source node] [source layer][target node][target layer] [weight]
    edge_cnt = 0
    file_name = "./muxViz-master/MultiGraph_Layer_edges_extended_{}-
{}.edges".format(author1,author2)
    with open(file_name, "w", encoding="utf-8") as file:
        for author_1, author_2 in Primary_edge_dict:
```

```
            _, weight, _ = Primary_edge_dict[tuple((author_1, author_2))]

            layerID_from = layer_dict["Authors"]
            layerID_to = layer_dict["Authors"]
            #Giving emphasis to primary author edges by increasing weight of primary author
edges by multiple of 4.
            row_str = "{} {} {} {} {}\n".format(index_dict[author_1],layerID_from,
                                                index_dict[author_2], layerID_to,
weight)
            file.write(row_str)
            edge_cnt += 1
        file.close()
    print(edge_cnt)

    print(edge_cnt, cnt)

    #Final recommendation pairs Configuration files to be imported to MuxViz as assembly
file.
    with open("./muxViz-master/MultiGraph_node_CONFIG_{}-{}.txt".format(author1,author2),
"w", encoding="utf-8") as file:
        edge_file = "MultiGraph_Layer_edges_extended_{}-{}.edges".format(author1,author2)
        layer_file = "MultiGraph_Layers_{}-{}.txt".format(author1,author2)
        node_layout = "MultiGraph_node_layout_{}-{}.txt".format(author1,author2)
        row_str = "{};{};{}\n".format(edge_file,layer_file, node_layout)
        file.write(row_str)
        file.close()
```

9.6 Homogenous (single-level) & Heterogeneous (multi-level) Research Network Visualisation

Stage six is based off python 3 scripts. In the final deployed model these would be called through a Rshiny Interface containing a reticulate R & Python application. When selected by users this would create the preferred output for all MuxViz Visualizations requested. Furthermore, automatically the Gephi File creation script will be run to create optional Gephi files ready for loading into a the Gephi Visualization Software outside of our Rshiny Application. Additional improvements could be providing users with option for either dynamic (temporal) year-by-year layers for assessing growth in research collaboration network & furthermore, adding in country, publication year, citation network and other research environment dimensions into visualisations.

Script 6.1 Preparation of .txt and .edge files for using MuxViz interactive overall homogenous co-authorship graph visualisation tool through Rshiny, incorporating all recommendations nodes and edges of stage five.

```
#import libraries:
import pickle
import pandas as pd
import matplotlib.pyplot as plt
from math import isclose
import os

#Getting primary author names:
primary_authors = pd.read_csv("C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/2-3 - Disambiguation and Network Comparison
(IGM)/Disambiguation_summary.csv")
primary_authors_set = set(primary_authors.Name)

#Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/5-Link
Prediction(Recommendations)/1-Preparing multitype nodes and edge
dataset/Hetergenous_dataset_final.pickle",'rb')
unpickler = pickle.Unpickler(file)
Hetero_dataset = unpickler.load()
Hetero_dataset = Hetero_dataset.loc[Hetero_dataset.weight > 1]
Hetero_dataset = Hetero_dataset[Hetero_dataset.relation == 'AUTHOR-(P)-AUTHOR']

#Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic
Representation/1-Secondary Author Pruning & Topic Modelling/Final_dataset.pickle",'rb')

#Final_dataset = pickle.load(file)
unpickler = pickle.Unpickler(file)
Author_dataset = unpickler.load()

#Creating different node types:
author_nodes = [author.replace(" ", "_") for author in set(Author_dataset["name"]) ]
author_nodes = set(author_nodes)
for coauthors in set(Author_dataset["coauthors"]):
    coauthors = coauthors.split(" ")
```

```
    for coauthor in coauthors:
        if not coauthor in author_nodes:
            author_nodes.add(coauthor)
author_nodes = pd.DataFrame(index=author_nodes)

import os
directory = "C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/6-Network
Vizualisation/2-Heterogeneous-Multiplex Vizualizations MuxVix File Creation"
os.chdir(directory + "/muxViz_network_files")
relations = {"Authors"}
layer_dict = dict()
cnt = 0
for relation in relations:
    cnt += 1
    layer_dict[relation] = cnt

with open("./muxViz-master/MultiGraph_Layers_COAUTHOR.txt", "w", encoding="utf-8") as file:
    file.write("layerID layerLabel\n")
    for layer in layer_dict.keys():
        row_str = "{} {}\n".format(layer_dict[layer], layer)
        print(row_str)
        file.write(row_str)
    file.close()


#Loading in Link Prediction results for recommendations:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/5-Link
Prediction(Recommendations)/2-Implementation of Metapath embedding and Link
Prediction/Hetergenous_Recommendations.pickle",'rb')
unpickler = pickle.Unpickler(file)
Recommendations_dts = unpickler.load()

#Removing recommendations where label predicted is negative (not recommended to
collaborate):
Recommendations_dts = Recommendations_dts[Recommendations_dts.Predicted_Label == 1]
#Looking for any matching coauthor pair who have been recommended to collaborate in the
development network dataset
#and do not yet share a co-author edge (i.e. have no coauthored a paper together and
therefore will not be in hetero-edge
#dataset).

#Building coauthor set for hetergenous dataset to check against for new collaboration
pairs:
coauthor_set = set()
for source, target, relation, weight in Hetero_dataset[Hetero_dataset.relation == 'AUTHOR-
(P)-AUTHOR'].values:
    if tuple((source, target)) not in coauthor_set:
        coauthor_set.add(tuple((source, target)))
print(len(coauthor_set), "unique coauthor pairs")
#note: will remove true label in final version as there will be no true label for current
pub year only predicted labels.
```

```
raw_cnt = 0
prob_filtered_cnt = 0
primary_recommendations = 0
secondary_recommendations = 0
primary_secondary_recmmendations_list = []
for pred_label, pred_prob, true_label, author1, author2 in Recommendations_dts.values:
    if (not tuple((author1,author2)) in coauthor_set) and (not tuple((author2,author1)) in
coauthor_set):
        raw_cnt += 1
        #print(pred_prob)
        if pred_prob > 0.00:
            prob_filtered_cnt += 1
            if ((author1 in primary_authors_set) or (author2 in primary_authors_set)) and
not ((author1 in primary_authors_set) and (author2 in primary_authors_set)):
                secondary_recommendations += 1
                primary_secondary_recommendations_list.append([pred_prob, author1, author2,
"secondary"])
            elif (author1 in primary_authors_set) and (author2 in primary_authors_set):
                    primary_recommendations += 1
                    primary_secondary_recommendations_list.append([pred_prob, author1,
author2, "primary"])

#Creating node density dictionaries to calculate standardised degree for each node- default
0 (no edges):
author_density_dict = dict()
for author in author_nodes.index:
    author_density_dict[author] = 0

for source, target, relation, weight in Hetero_dataset.values:
    if source.replace(" ", "_") in author_density_dict:
        author_density_dict[source.replace(" ", "_")] += 1
    if target.replace(" ", "_") in author_density_dict:
        author_density_dict[target.replace(" ", "_")] += 1
author_node_length = len(author_nodes.index)

#Standardizing each node type degree, separately by node type:
for author in author_density_dict:
    author_density_dict[author] = author_density_dict[author]/(author_node_length-1)


import numpy as np
percentile = 99
#Taking {percentile} as threshold cut-off value for network nodes and edges:
author_density_threshold = np.percentile(list(author_density_dict.values()), percentile)
max_weight = max(list(Hetero_dataset["weight"]))
#Normalizing edge weights:
Hetero_dataset["weight"] = Hetero_dataset["weight"]/max_weight
average_edge_weight = np.mean(Hetero_dataset["weight"])
edge_weight_threshold = np.percentile(list(Hetero_dataset["weight"]),percentile)
max_weight_normalized = max(list(Hetero_dataset["weight"]))
```

```
#Creating recommendation set for checking authors:
auth_recommended_dict = dict()
auth_recommended_dict["primary"] = set()
auth_recommended_dict["secondary"] = set()
auth_recommended_dict["primary_pairs"] = set()
auth_recommended_dict["secondary_pairs"] = set()

for prob, author1, author2, rec_type in primary_secondary_recommendations_list:
    if rec_type == "primary":
        auth_recommended_dict["primary"].add(author1)
        auth_recommended_dict["primary"].add(author2)
        auth_recommended_dict["primary_pairs"].add(tuple((author1, author2)))

    elif rec_type == "secondary":
        auth_recommended_dict["secondary"].add(author1)
        auth_recommended_dict["secondary"].add(author2)
        auth_recommended_dict["secondary_pairs"].add(tuple((author1, author2)))

Primary_author_dict = dict()
Primary_edge_dict = dict()
for source, target, relation, weight in Hetero_dataset.values:
        if ((source in primary_authors_set ) or (target in primary_authors_set)):
            #Checking if author is a primary researcher or not:
            if source in primary_authors_set and not source in Primary_author_dict:
                Primary_author_dict[source] = True
            elif not source in primary_authors_set and not source in Primary_author_dict:
                Primary_author_dict[author] = False
            if target in primary_authors_set and not target in Primary_author_dict:
                Primary_author_dict[target] = True
            elif not target in primary_authors_set and not target in Primary_author_dict:
                Primary_author_dict[target] = False
            #Checking if a primary edge or not:
            if not tuple((source, target)) in Primary_edge_dict and not tuple((target,
source)) in Primary_edge_dict:
                Primary_edge_dict[tuple((source, target))] = (True,weight,False)
            else:
                print("EXCEPTION-4",(tuple((source, target)) in Primary_edge_dict or
tuple((target, source)) in Primary_edge_dict ))
        elif ((not source in primary_authors_set) and (not target in primary_authors_set)):
            Primary_author_dict[source] = False
            Primary_author_dict[target] = False
            if not tuple((source, target)) in Primary_edge_dict and not tuple((target,
source)) in Primary_edge_dict:
                Primary_edge_dict[tuple((source, target))] = (False,weight, False)
        else:
            print("EXCEPTION5",source, target, relation, weight)
for author1,author2 in auth_recommended_dict["primary_pairs"]:
    #Primary authors already put into dictionary, thereby can assume if not in Primary dict
than is a secondary author.
    if not author1 in Primary_author_dict:
```

```
        Primary_author_dict[author1] = False
    if not author2 in Primary_author_dict:
        Primary_author_dict[author2] = False
    if not tuple((author1,author2)) in Primary_edge_dict and not tuple((author2,author1))
in Primary_edge_dict:
        Primary_edge_dict[tuple((author1,author2))] = (True, edge_weight_threshold,
"primary_pairs")

for author1,author2 in auth_recommended_dict["secondary_pairs"]:
    #All authors in secondary pair are secondary authors.
    if not author1 in Primary_author_dict:
        Primary_author_dict[author1] = False
    if not author2 in Primary_author_dict:
        Primary_author_dict[author2] = False
    if not tuple((author1,author2)) in Primary_edge_dict and not tuple((author2,author1))
in Primary_edge_dict:
        Primary_edge_dict[tuple((author1,author2))] = (True, edge_weight_threshold,
"secondary_pairs")

index_count = 0
index_dict = dict()
with open("./muxViz-master/MultiGraph_node_layout_COAUTHOR.txt", "w", encoding="utf-8") as
file:
    file.write("nodeID nodeLabel\n")
    for author in Primary_author_dict:
            index_count += 1
            index_dict[author] = index_count
            row_str = "{} {}\n".format(index_dict[author], author)
            print(row_str)
            file.write(row_str)
    file.close()
print(index_count)

#Creating node colour and size file - NEW version only primary nodes and non-primary nodes
above threshold for degree:
cnt = 0
recom_author_cnt = 0
with open("./muxViz-master/MultiGraph_node_colour_size_COAUTHOR.txt", "w", encoding="utf-
8") as file:
    file.write("nodeID layerID color size\n")
    for node in Primary_author_dict:

        if (node in primary_authors_set) and (not (node in
auth_recommended_dict["secondary"])) and (not (node in auth_recommended_dict["primary"])):
                    row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Authors"], "orange",author_density_dict[node]*5)
                    print(row_str)
                    cnt += 1
                    file.write(row_str)
        elif (node in primary_authors_set) and ((node in
auth_recommended_dict["secondary"]) or ( node in auth_recommended_dict["primary"])):
```

```
                    print("Recommendation primary node:", node)
                    row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Authors"], "red",author_density_dict[node]*5)
                    print(row_str)
                    cnt += 1
                    file.write(row_str)
        elif (not node in primary_authors_set) and ((node in
auth_recommended_dict["secondary"]) or (node in auth_recommended_dict["primary"])):
                    print("Recommendation secondary node:", node)
                    row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Authors"], "purple",(author_density_dict[node] + author_density_threshold)*5)
                    print(row_str)
                    cnt += 1
                    file.write(row_str)
        else:
                    row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Authors"], "green",author_density_dict[node]*5)
                    print(row_str)
                    cnt += 1
                    file.write(row_str)

    file.close()



#Creating edge size and colour file:
#FORMAT: nodeID.from layerID.from nodeID.to layerID.to color size
cnt = 0
with open("./muxViz-master/MultiGraph_edge_colour_size_COAUTHOR.txt", "w", encoding="utf-
8") as file:
    file.write("nodeID.from layerID.from nodeID.to layerID.to color size\n")
    for author1, author2 in Primary_edge_dict:
        primary_flag, weight, pair_bool = Primary_edge_dict[tuple((author1, author2))]
        #Giving emphasis in primary author edges in Vizualisation making edges different
colours and of significantly higher weight:
        #Looking for primary author edge instances:
        if  primary_flag == True:
            if pair_bool == "primary_pairs":
                #This is a primary recommendation pair edge between two primary authors
(Blue):
                layerID_from = layer_dict["Authors"]
                layerID_to = layer_dict["Authors"]
                row_str = "{} {} {} {} {} {}\n".format(index_dict[author1],layerID_from,
                                                       index_dict[author2], layerID_to,
                                                       "blue", weight)
                file.write(row_str)
                cnt += 1
                print("primary_pair edge")
            elif pair_bool == "secondary_pairs":
                #This is a secondary recommendation pair edge with a primary author
(Purple):
```

```
                layerID_from = layer_dict["Authors"]
                layerID_to = layer_dict["Authors"]
                row_str = "{} {} {} {} {} {}\n".format(index_dict[author1],layerID_from,
                                                       index_dict[author2], layerID_to,
                                                       "red", weight)
                file.write(row_str)
                cnt += 1
                print("secondary_pair edge")
            else: #pair_bool == False NOT a recommendation edge, although IS a normal
primary edge:
                #This is a normal primary edge (ORANGE):
                layerID_from = layer_dict["Authors"]
                layerID_to = layer_dict["Authors"]
                row_str = "{} {} {} {} {} {}\n".format(index_dict[author1],layerID_from,
                                                       index_dict[author2], layerID_to,
                                                       "orange", weight)
                file.write(row_str)
                cnt += 1

        elif  primary_flag == False:
            cnt += 1 #Not a primary edge, thereby edge will take on default secondary
colour set in MuxViz.
        else:
            print("EXCEPTION: Edge Colour & Size")



    file.close()
print("{} edges".format(cnt))


#[source node] [source layer][target node][target layer] [weight]
edge_cnt = 0
file_name = "./muxViz-master/MultiGraph_Layer_edges_extended_COAUTHOR.edges"
with open(file_name, "w", encoding="utf-8") as file:
    for author1, author2 in Primary_edge_dict:
        _, weight, _ = Primary_edge_dict[tuple((author1, author2))]

        layerID_from = layer_dict["Authors"]
        layerID_to = layer_dict["Authors"]
        #Giving emphasis to primary author edges by increasing weight of primary author
edges by multiple of 4.
        row_str = "{} {} {} {} {}\n".format(index_dict[author1],layerID_from,
                                                index_dict[author2], layerID_to, weight)
        file.write(row_str)
        edge_cnt += 1
    file.close()
print(edge_cnt)

print(edge_cnt, cnt)
```

```
#Creating Final Configuration file to be imported to MuxVet as assembly file.
with open("./muxViz-master/MultiGraph_node_CONFIG_COAUTHOR.txt", "w", encoding="utf-8") as
file:
    edge_file = "MultiGraph_Layer_edges_extended_COAUTHOR.edges"
    layer_file = "MultiGraph_Layers_COAUTHOR.txt"
    node_layout = "MultiGraph_node_layout_COAUTHOR.txt"
    row_str = "{};{};{}\n".format(edge_file,layer_file, node_layout)
    file.write(row_str)
    file.close()
```

Script 6.2 Gephi File Preparation for Users to use Gephi Visualization Software outside of our toolkit.

```
#Import libraries:
import pickle
import pandas as pd
import networkx as nx
import pandas as pd

#Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/5-Link
Prediction(Recommendations)/1-Preparing multitype nodes and edge
dataset/Hetergenous_dataset_final.pickle",'rb')
unpickler = pickle.Unpickler(file)
Hetero_dataset = unpickler.load()

#Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic
Representation/1-Secondary Author Pruning & Topic Modelling/Final_dataset.pickle",'rb')
unpickler = pickle.Unpickler(file)
Author_dataset = unpickler.load()

#Getting primary author names:
primary_authors = pd.read_csv("C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/2-3 - Disambiguation and Network Comparison
(IGM)/Disambiguation_summary.csv")
primary_authors_set = set(primary_authors.Name)

#Creating different node types:
org_nodes = pd.DataFrame(index=[org for org in set(Author_dataset["organization"]) if org
!= "NA"])
author_nodes = [author for author in set(Author_dataset["name"]) if author != "NA"]
author_nodes = set(author_nodes)
for coauthors in set(Author_dataset["coauthors"]):
    coauthors = coauthors.split(" ")
    for coauthor in coauthors:
        if not coauthor in author_nodes:
            author_nodes.add(coauthor)
author_nodes = pd.DataFrame(index=author_nodes)

topic_set = set()
for topic_list in Author_dataset["TOPICS"]:
    for topic, prop in topic_list:
        topic_set.add("T{}".format(topic))
topic_nodes =  pd.DataFrame(index=topic_set)
venue_nodes = pd.DataFrame(index=[jconf for jconf in set(Author_dataset["jconf"]) if jconf
!= "NA"])

#Creating homogenous networkx Author < TOPIC > Author network for Gephi vizualisation:
#author_topicNet = Hetero_dataset[Hetero_dataset.relation == "AUTHOR-(T)-AUTHOR"]
#author_topicNetG = nx.Graph()
```

```
#author_topicNetG.add_nodes_from(author_nodes.index)
#author_topicNetG.add_edges_from([(source,target,{"weight": weight}) for source, target,
relation, weight in author_topicNet.values if weight >= 1])
#nx.write_gexf(author_topicNetG, "author_topic_network.gexf")

import os
directory = "C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/6-Network
Vizualisation/1-Homogeneous-Single Relation Vizualisation GEPHI File Creation"
os.chdir(directory + "/GEPHI_network_files")

#Creating homogenous networkx Author < PAPER > Author network for Gephi vizualisation:
author_coauthorNet = Hetero_dataset[Hetero_dataset.relation == "AUTHOR-(P)-AUTHOR"]
author_coauthorNetG = nx.Graph()
author_coauthorNetG.add_nodes_from(author_nodes.index)
author_coauthorNetG.nodes()
author_coauthorNetG.add_edges_from([(source,target,{"weight": weight}) for source, target,
relation, weight in author_coauthorNet.values])
nx.write_gexf(author_coauthorNetG, "author_coauthor_network.gexf")


#Reformatting author nodes for primary authors only:
author_nodes = [author for author in set(Author_dataset["name"]) if author != "NA" and
author in primary_authors_set]
author_nodes = set(author_nodes)
for coauthors in set(Author_dataset["coauthors"]):
    coauthors = coauthors.split(" ")
    for coauthor in coauthors:
        if not coauthor in author_nodes and coauthor in primary_authors_set:
            author_nodes.add(coauthor)
author_nodes = pd.DataFrame(index=author_nodes)


#Creating homogenous networkx ORG < AUTHOR > ORG network for Gephi vizualisation:
org_orgNet = Hetero_dataset[Hetero_dataset.relation == "ORG-(A)-ORG"]
org_orgNetG = nx.Graph()
org_orgNetG.add_nodes_from(org_nodes.index)
org_orgNetG.nodes()
org_orgNetG.add_edges_from([(source,target,{"weight": weight}) for source, target,
relation, weight in org_orgNet.values])
nx.write_gexf(org_orgNetG, "org_org_network.gexf")

#Creating homogenous networkx ORG < PAPER > VENUE network for Gephi vizualisation:
#org_venueNet = Hetero_dataset[Hetero_dataset.relation == "ORG-(P)-VENUE"]
#org_venueNetG = nx.Graph()
#org_venueNetG.add_nodes_from(org_nodes.index)
#org_venueNetG.add_nodes_from(venue_nodes.index)
#org_venueNetG.nodes()
#org_venueNetG.add_edges_from([(source,target,{"weight": weight}) for source, target,
relation, weight in org_venueNet.values])
#nx.write_gexf(org_venueNetG, "org_venue_network.gexf")
```

```
#Creating homogenous networkx AUTHOR < PAPER > VENUE network for Gephi visualisation:
#author_venueNet = Hetero_dataset[Hetero_dataset.relation == "AUTHOR-(P)-VENUE"]
#author_venueNetG = nx.Graph()
#author_venueNetG.add_nodes_from(author_nodes.index)
#author_venueNetG.add_nodes_from(venue_nodes.index)
#author_venueNetG.nodes()
#author_venueNetG.add_edges_from([(source,target,{"weight": weight}) for source, target,
relation, weight in author_venueNet.values])
#nx.write_gexf(author_venueNetG, "author_venue_network.gexf")

#Creating homogenous networkx AUTHOR < PAPER > ORG network for Gephi vizualisation:
#author_orgNet = Hetero_dataset[Hetero_dataset.relation == "AUTHOR-(P)-ORG"]
#author_orgNetG = nx.Graph()
#author_orgNetG.add_nodes_from(org_nodes.index)
#author_orgNetG.add_nodes_from(author_nodes.index)
#author_orgNetG.nodes()
#author_orgNetG.add_edges_from([(source,target,{"weight": weight}) for source, target,
relation, weight in author_orgNet.values])
#nx.write_gexf(author_orgNetG, "author_org_network.gexf")

#Creating homogenous networkx AUTHOR < PAPER > TOPIC network for Gephi vizualisation:
#author_topicNet = Hetero_dataset[Hetero_dataset.relation == "AUTHOR-(P)-TOPIC"]
#author_topicNetG = nx.Graph()
#author_topicNetG.add_nodes_from(topic_nodes.index)
#author_topicNetG.add_nodes_from(author_nodes.index)
#author_topicNetG.nodes()
#author_topicNetG.add_edges_from([(source,target,{"weight": weight}) for source, target,
relation, weight in author_topicNet.values])
#nx.write_gexf(author_topicNetG, "author_topic_network.gexf")

#Creating homogenous networkx ORG < PAPER > TOPIC network for Gephi visualisation:
#org_topicNet = Hetero_dataset[Hetero_dataset.relation == "ORG-(P)-TOPIC"]
#org_topicNetG = nx.Graph()
#org_topicNetG.add_nodes_from(org_nodes.index)
#org_topicNetG.add_nodes_from(topic_nodes.index)
#org_topicNetG.nodes()
#org_topicNetG.add_edges_from([(source,target,{"weight": weight}) for source, target,
relation, weight in org_topicNet.values])
#nx.write_gexf(org_topicNetG, "author_topic_network.gexf")

#Creating homogenous networkx ORG < PAPER > VENUE network for Gephi visualisation:
#org_venueNet = Hetero_dataset[Hetero_dataset.relation == "ORG-(P)-VENUE"]
#org_venueNetG = nx.Graph()
#org_venueNetG.add_nodes_from(org_nodes.index)
#org_venueNetG.add_nodes_from(venue_nodes.index)
#org_venueNetG.nodes()
#org_venueNetG.add_edges_from([(source,target,{"weight": weight}) for source, target,
relation, weight in org_venueNet.values])
#nx.write_gexf(org_venueNetG, "org_venue_network.gexf")
```

Script 6.3 Preparation of Heterogeneous Multi-level or Multi-layer Independent Research Collaboration Network Visualization for Research Organisation

```
#Import libraries:
import pickle
import pandas as pd
import matplotlib.pyplot as plt
from math import isclose
import os

#Getting primary author names:
primary_authors = pd.read_csv("C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/2-3 - Disambiguation and Network Comparison
(IGM)/Disambiguation_summary.csv")
primary_authors_set = set(primary_authors.Name)

#Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/5-Link
Prediction(Recommendations)/1-Preparing multitype nodes and edge
dataset/Hetergenous_dataset_final.pickle",'rb')
unpickler = pickle.Unpickler(file)
Hetero_dataset = unpickler.load()
Hetero_dataset = Hetero_dataset.loc[Hetero_dataset.weight > 1]


#Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic
Representation/2-Assignment of Topic Labels using Modelled
Topics/Inferred_topic_label_dict.pickle",'rb')
unpickler = pickle.Unpickler(file)
Inferred_topic_label_dict = unpickler.load()

#Loading in Link Prediction results for recommendations:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/5-Link
Prediction(Recommendations)/2-Implementation of Metapath embedding and Link
Prediction/Hetergenous_Recommendations.pickle",'rb')
unpickler = pickle.Unpickler(file)
Recommendations_dts = unpickler.load()

#1. Looking for recommendations where, co-authors had NOT previously collaborated (no
AUTHOR<-PAPER->AUTHOR EDGE)
 #, both of these authors is a primary organisation author (or if not available secondly
just one is primary author
 # and thirdly, the predicted label/probability for the label is 1 and preferably greater
then 80% likelihood of
 #authors collaborating.


#2. NOTE: THIS IS ONLY A PRACTICE PIPELINE AS IT IS USING THE TEST DATA RESULTS, instead of
results of developed model.
#Extracted time variable for papers published, can be used in future for train/test on
years prior saving latest dataset year
#(current) for recommendations! - using dynamic rather than static graph.

#Removing recommendations where label predicted is negative (not recommended to
collaborate):
Recommendations_dts = Recommendations_dts[Recommendations_dts.Predicted_Label == 1]
```

```
#Looking for any matching coauthor pair who have been recommended to collaborate in the
development network dataset
#and do not yet share a co-author edge (i.e. have no coauthored a paper together and
therefore will not be in hetero-edge
#dataset).

#Building coauthor set for heterogeneous dataset to check against for new collaboration
pairs:
coauthor_set = set()
for source, target, relation, weight in Hetero_dataset[Hetero_dataset.relation == 'AUTHOR-
(P)-AUTHOR'].values:
    if tuple((source, target)) not in coauthor_set:
        coauthor_set.add(tuple((source, target)))
print(len(coauthor_set), "unique coauthor pairs")
#note: will remove true label in final version as there will be no true label for current
pub year only predicted labels.
raw_cnt = 0
prob_filtered_cnt = 0
primary_recommendations = 0
secondary_recommendations = 0

primary_secondary_recommendations_list = []
for pred_label, pred_prob, true_label, author1, author2 in Recommendations_dts.values:
    if (not tuple((author1,author2)) in coauthor_set) and (not tuple((author2,author1)) in
coauthor_set):
        raw_cnt += 1
        #print(pred_prob)
        if pred_prob > 0:
            prob_filtered_cnt += 1
            if ((author1 in primary_authors_set) or (author2 in primary_authors_set)) and
not ((author1 in primary_authors_set) and (author2 in primary_authors_set)):
                secondary_recommendations += 1
                primary_secondary_recommendations_list.append([pred_prob, author1, author2,
"secondary"])
            elif (author1 in primary_authors_set) and (author2 in primary_authors_set):
                    primary_recommendations += 1
                    primary_secondary_recommendations_list.append([pred_prob, author1,
author2, "primary"])

print("There were", raw_cnt,"instances of two authors being recommended (predicted) to
collaborate in this current year.")
print("\n")
print("Meanwhile, there were", prob_filtered_cnt,"instances of two authors being
recommended (predicted) to \
      collaborate in this current year with a high predicted probability of greater than
80%")
print("\n")
print("Of this", prob_filtered_cnt, "there was", secondary_recommendations, "secondary
recommendations (recommendations\
 between a primary author and a secondary author) and there was", primary_recommendations,
"primary recommendations\
(recommendations between two primary authors)")

#Loading in cleaned and topic-modelled researcher dataset by paper per researcher:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/4-Topic
Representation/1-Secondary Author Pruning & Topic Modelling/Final_dataset.pickle",'rb')
#Final_dataset = pickle.load(file)
unpickler = pickle.Unpickler(file)
Author_dataset = unpickler.load()

#Creating different node types:
```

```
org_nodes = pd.DataFrame(index=[org.replace(" ", "_") for org in
set(Author_dataset["organization"]) if org != "palliative medicine"])
author_nodes = [author.replace(" ", "_") for author in set(Author_dataset["name"]) ]
author_nodes = set(author_nodes)
for coauthors in set(Author_dataset["coauthors"]):
    coauthors = coauthors.split(" ")
    for coauthor in coauthors:
        if not coauthor in author_nodes:
            author_nodes.add(coauthor)
author_nodes = pd.DataFrame(index=author_nodes)

venue_nodes = pd.DataFrame(index=[jconf.replace(" ", "_") for jconf in
set(Author_dataset["jconf"]) if not jconf in org_nodes.index])

topic_set = set()
for topic_list in Author_dataset["TOPICS"]:
    for topic, prop in topic_list:
        topic_set.add("T{}".format(str(int(topic) + 1)))
topic_nodes =  pd.DataFrame(index=topic_set)

import os
directory = "C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/6-Network
Vizualisation/2-Heterogeneous-Multiplex Vizualizations MuxVix File Creation"
os.chdir(directory + "/muxViz_network_files")



relations = {"Topics","Organisations","Authors", "Venues(Journal/Conferences)"}


layer_dict = dict()
cnt = 0
for relation in relations:
    cnt += 1
    layer_dict[relation] = cnt

with open("./muxViz-master/MultiGraph_Layers.txt", "w", encoding="utf-8") as file:
    file.write("layerID layerLabel\n")
    for layer in layer_dict.keys():
        row_str = "{} {}\n".format(layer_dict[layer], layer)
        print(row_str)
        file.write(row_str)
    file.close()

#layerID layerLabel
#1 L1
#2 L2
#3 L3
#4 L4
#5 L5
#6 L6

#Extended edges list
#An extended edges list is a new format that allows to specify all possible types of links,
#intra- and inter-layer. Each line specifies the source node (first column) and
#the source layer (second column), the destination node (third column) and
#the destination layer (fourth column), possibly weighted by an integer or floating number
(fifth column).

#Creating node density dictionaries to calculate standardized degree for each node- default
0 (no edges):
```

```
author_density_dict = dict()
org_density_dict = dict()
venue_density_dict = dict()
topic_density_dict = dict()


#Automatic-cleaning list (rather than user-input for cleaning below shared orgs names)
Correct_primary_org_name = "University_of_New_South_Wales"
#Creating UNSW dictionary to clean UNSW organisation names on user-request:
UNSW_dict = {"cnr":"Consiglio_Nazionale_delle_Ricerche-
NRC","ingham_inst_appl_med_res":"ingham_institute_for_applied_medical_research","harvard_me
d_sch":"harvard_univ","sydney_childrens_hosp_network":"sydney_childrens_hosp","liverpool_hl
th_serv":"liverpool_hosp","univ_western_sydney":"western_sydney_univ","university_of_wester
n_sydney_-
_campbelltown_campus":"western_sydney_univ","unsw_sydney":Correct_primary_org_name,"unsw_au
stralia":Correct_primary_org_name,"unsw":Correct_primary_org_name,

"univ_new_south_wales":Correct_primary_org_name,"university_of_new_south_wales_(unsw)_austr
alia":Correct_primary_org_name,
             "univ_new_s_wales":Correct_primary_org_name,
"univ_nsw":Correct_primary_org_name, "univ new south wales
sydney":Correct_primary_org_name}
user_clean = False
while user_clean == True:
    request = input("Do you want to clean and merge orgs to single name, type (yes/no)\n")
    if request.lower() == "yes":
        clean = input("Type incorrect org name instance to remove (include _ instead of
spaces\n")
        merge = input("Type correct org name to merge incorrect name instance with (include
_ instead of spaces\n")
        UNSW_dict[clean] = merge
    elif request.lower() == "no":
        user_clean = False
#Automatic-cleaning list (rather than user-input for cleaning below shared venue names)
#Creating Venue cleaning dictionary to clean venue names on user-request:
Venue_clean_dict = {"clinical_oncology":"journal_of_clinical_oncology",
                    "palliative_medicine":"journal_of_palliative_medicine",

"radiotherapy_and_oncology":"international_journal_of_radiation_oncology_biology_physics"}
user_clean = False
while user_clean == True:
    request = input("Do you want to clean and merge venues to single name, type
(yes/no)\n")
    if request.lower() == "yes":
        clean = input("Type incorrect venue name instance to remove (include _ instead of
spaces\n")
        merge = input("Type correct venue name to merge incorrect name instance with
(include _ instead of spaces\n")
        Venue_clean_dict[clean] = merge
    elif request.lower() == "no":
        user_clean = False

for author in author_nodes.index:
    author_density_dict[author] = 0
for org in org_nodes.index:
    if not org in UNSW_dict:
        org_density_dict[org] = 0
    if org in UNSW_dict and not UNSW_dict[org] in org_density_dict:
        org_density_dict[UNSW_dict[org]] = 0
for venue in venue_nodes.index:
    if not venue in Venue_clean_dict:
```

```
        venue_density_dict[venue] = 0
    if venue in Venue_clean_dict and not Venue_clean_dict[venue] in venue_density_dict:
        venue_density_dict[Venue_clean_dict[venue]] = 0
for topic in topic_nodes.index:
    topic_density_dict[topic] = 0
for source, target, relation, weight in Hetero_dataset.values:
    if source.replace(" ", "_") in author_density_dict:
        author_density_dict[source.replace(" ", "_")] += 1
    elif source.replace(" ", "_") in org_density_dict or source.replace(" ", "_") in
UNSW_dict:
        if source.replace(" ", "_") in org_density_dict :
            org_density_dict[source.replace(" ", "_")] += 1
        elif source.replace(" ", "_") in UNSW_dict:
            #manual fix for org <-> venue error:
            #if org_density_dict[UNSW_dict[source.replace(" ", "_")]] ==
"journal_of_palliative_medicine":
            #    continue
            #aggregating all UNSW org names to same org.
            #manual fix for org <-> venue error:
            #elif not org_density_dict[UNSW_dict[source.replace(" ", "_")]] ==
"journal_of_palliative_medicine":
            org_density_dict[UNSW_dict[source.replace(" ", "_")]]+= 1
    elif source.replace(" ", "_") in venue_density_dict  or source.replace(" ", "_") in
Venue_clean_dict:
        if source.replace(" ", "_") in venue_density_dict :
            venue_density_dict[source.replace(" ", "_")] += 1
        elif source.replace(" ", "_") in Venue_clean_dict:
            #aggregating all venue names to same correct venue name.
             venue_density_dict[Venue_clean_dict[source.replace(" ", "_")]]+= 1

    elif source.replace(" ", "_") in topic_density_dict:
        topic_density_dict[source.replace(" ", "_")] += 1

    if target.replace(" ", "_") in author_density_dict:
        author_density_dict[target.replace(" ", "_")] += 1
    elif target.replace(" ", "_") in org_density_dict or target.replace(" ", "_") in
UNSW_dict:
        if target.replace(" ", "_") in org_density_dict :
            org_density_dict[target.replace(" ", "_")] += 1
        elif target.replace(" ", "_") in UNSW_dict:
            #aggregating all UNSW org names to same org.
             org_density_dict[UNSW_dict[target.replace(" ", "_")]]+= 1
    elif target.replace(" ", "_") in venue_density_dict or target.replace(" ", "_") in
Venue_clean_dict:
        if target.replace(" ", "_") in venue_density_dict :
            venue_density_dict[target.replace(" ", "_")] += 1
        elif target.replace(" ", "_") in Venue_clean_dict:
            #aggregating all venue names to same correct venue name.
             venue_density_dict[Venue_clean_dict[target.replace(" ", "_")]]+= 1
    elif target.replace(" ", "_") in topic_density_dict:
        topic_density_dict[target.replace(" ", "_")] += 1
#manually correcting an error in orgs and venues - where palliative care journal is both a
org and venue!

#Prior to normalizing node densities, creating greatest influencers/nodes table for
Visualization section:
author_list = []
for author in author_density_dict:
    author_list.append([author, author_density_dict[author]])
#Creating author dataframe to sort authors by:
author_df = pd.DataFrame(author_list, columns = ["Researcher","Unnormalized Density"])
```

```
author_df.sort_values(by=["Unnormalized Density"], ascending=False, inplace=True)
#Taking top-50 Venues and saving as csv file:
author_df = author_df[:50]
author_df.to_csv("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/6-Network
Vizualisation/Top-50-Influential-Researchers.csv")


#Prior to normalizing node densities, creating greatest influencers/nodes table for
Visualization section:
venue_list = []
for venue in venue_density_dict:
    venue_list.append([venue.replace("_", " "), venue_density_dict[venue]])
#Creating venue dataframe to sort venues by:
venue_df = pd.DataFrame(venue_list, columns = ["Venue","Unnormalized Density"])
venue_df.sort_values(by=["Unnormalized Density"], ascending=False, inplace=True)
#Taking top-50 Venues and saving as csv file:
venue_df = venue_df[:50]
venue_df.to_csv("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/6-Network
Visualization/Top-50-Influential-Venues.csv")

#Prior to normalizing node density, creating greatest influencers/nodes table for
Visualization section:
topic_list = []
for topic in topic_density_dict:
    topic_list.append([Inferred_topic_label_dict[topic], topic_density_dict[topic]])
#Creating venue dataframe to sort venues by:
topic_df = pd.DataFrame(topic_list, columns = ["Topic","Unnormalized Density"])
topic_df.sort_values(by=["Unnormalized Density"], ascending=False, inplace=True)
#Taking Topics ordered by most influencial network topics and saving as csv file:
topic_df = topic_df[:len(topic_df)]
topic_df.to_csv("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/6-Network
Vizualisation/Top-50-Influential-Topics.csv")

#Prior to normalizing node density, creating greatest influencers/nodes table for
Visualization section:
org_list = []
for org in org_density_dict:
    org_list.append([org.replace("_", " "), org_density_dict[org]])
#Creating org dataframe to sort orgs by:
org_df = pd.DataFrame(org_list, columns = ["Organisation","Unnormalized Density"])
org_df.sort_values(by=["Unnormalized Density"], ascending=False, inplace=True)
#Taking top-50 Organisations and saving as csv file:
org_df = org_df[:50]
org_df.to_csv("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/6-Network
Vizualisation/Top-50-Influential-Organisations.csv")




author_node_length = len(author_nodes.index)
org_node_length = len(org_nodes.index)
venue_node_length = len(venue_nodes.index)
topic_node_length = len(topic_nodes.index)
total_node_length = author_node_length+org_node_length+venue_node_length+topic_node_length
#Standardizing each node type degree, seperately by node type:
for author in author_density_dict:
    author_density_dict[author] = round(author_density_dict[author]/(total_node_length-
1),4)
```

```
for org in org_density_dict:
    org_density_dict[org] = round(org_density_dict[org]/(total_node_length-1),4)
for venue in venue_density_dict:
    venue_density_dict[venue] = round(venue_density_dict[venue]/(total_node_length-1),4)
for topic in topic_density_dict:
    topic_density_dict[topic] = round(topic_density_dict[topic]/(total_node_length-1),4)

#Checking for best cut-off for showing non-primary author nodes in visualization:
#from matplotlib import pyplot as plt
plt.hist(author_density_dict.values()) #0.05 * 100
plt.hist(venue_density_dict.values()) #>  #0.01  *100
plt.hist(org_density_dict.values())  #> 0.005  *100
plt.hist(topic_density_dict.values())  #> 0.02  *100

import numpy as np

#Taking 30% percentile as threshold cut-off value:
venue_density_threshold = np.percentile(list(venue_density_dict.values()), 96)
org_density_threshold = np.percentile(list(org_density_dict.values()), 97)
author_density_threshold = np.percentile(list(author_density_dict.values()), 80)
topic_density_threshold = np.percentile(list(topic_density_dict.values()), 0)  #include all
topic nodes.


#Checking threshold for keeping edges:
plt.hist(list(Hetero_dataset["weight"]))  #10
max_weight = max(list(Hetero_dataset["weight"]))
#Normalizing edge weights:
Hetero_dataset["weight"] = Hetero_dataset["weight"]/max_weight
edge_weight_threshold = np.percentile(list(Hetero_dataset["weight"]), 50)

Primary_venue_dict = dict()
Primary_org_dict = dict()
Primary_author_dict = dict()
Primary_topic_dict = dict()

Hetero_primary_flag_dts = []
for source, target, relation, weight in Hetero_dataset.values:
    primary_edge = False
    for org in org_nodes.index:
        if ((org == source.replace(" ", "_") or org == target.replace(" ", "_")) and
            ((source in primary_authors_set ) or (target in primary_authors_set))):
            if not org in UNSW_dict:
                Primary_org_dict[org] = True
                primary_edge = True
                #print("TRUE")
            elif org in UNSW_dict:
                Primary_org_dict[UNSW_dict[org]] = True
                primary_edge = True
    for venue in venue_nodes.index:
        if ((venue == source.replace(" ", "_") or venue == target.replace(" ", "_")) and
            ((source in primary_authors_set ) or (target in primary_authors_set))):
            if not venue in Venue_clean_dict:
                Primary_venue_dict[venue] = True
                primary_edge = True
                #print("TRUE")
            elif venue in Venue_clean_dict:
                Primary_venue_dict[Venue_clean_dict[venue]] = True
                primary_edge = True

    for topic in topic_nodes.index:
```

```
        if ((topic == source.replace(" ", "_") or topic == target.replace(" ", "_")) and
            ((source in primary_authors_set ) or (target in primary_authors_set))):
            Primary_topic_dict[topic] = True
            #print("TRUE")
            primary_edge = True

    for author in author_nodes.index:
        if ((author == source.replace(" ", "_") or author == target.replace(" ", "_")) and
            ((source in primary_authors_set ) or (target in primary_authors_set))):
            if author_density_dict[author] > 0:
#NOTE: INCLUDING THIS TO REMOVE RECOMMENDED & VIZUALISED 0 edge (isolated nodes) which
should not exist due to co-author relations, implying a bug.
                Primary_author_dict[author] = True
                #print("TRUE")
                primary_edge = True
    if primary_edge == True:
        Hetero_primary_flag_dts.append([source, target,relation, weight, True])
    elif primary_edge == False:  #Not a primary edge, however, checking if above threshold
for non-primary nodes:
        #trying to capture relationships between organisations.
        if relation == 'ORG-(A)-ORG':
            if weight >= 0.03:
                Hetero_primary_flag_dts.append([source, target,relation, weight, False])
        elif weight >= 0.1:
            Hetero_primary_flag_dts.append([source, target,relation, weight, False])
#Not including non-primary edges or non-primary nodes.

for author in author_nodes.index:
    if author not in Primary_author_dict:
        Primary_author_dict[author] = False
for topic in topic_nodes.index:
    if topic not in Primary_topic_dict:
        Primary_topic_dict[topic] = False
for venue in venue_nodes.index:
    if not venue  in Primary_venue_dict and not venue in Venue_clean_dict:
        Primary_venue_dict[venue] = False
    elif venue in Venue_clean_dict:
        if not Venue_clean_dict[venue] in Primary_venue_dict:
            Primary_venue_dict[Venue_clean_dict[venue]] = False
for org in org_nodes.index:
    if not org  in Primary_org_dict and not org in UNSW_dict:
        Primary_org_dict[org] = False
    elif org in UNSW_dict:
        if not UNSW_dict[org] in Primary_org_dict:
            Primary_org_dict[UNSW_dict[org]] = False

global_primary_dict = dict()
for author in Primary_author_dict:
    global_primary_dict[author] = Primary_author_dict[author]
for topic in Primary_topic_dict:
    global_primary_dict[topic] = Primary_topic_dict[topic]
for venue in Primary_venue_dict:
    global_primary_dict[venue] = Primary_venue_dict[venue]
for org in Primary_org_dict:
    global_primary_dict[org] = Primary_org_dict[org]


#Creating recomm set for checking authors:
auth_recommended_dict = dict()
auth_recommended_dict["primary"] = set()
auth_recommended_dict["secondary"] = set()
```

```
auth_recommended_dict["primary_pairs"] = set()
auth_recommended_dict["secondary_pairs"] = set()
auth_recommended_dict["general_pairs"] = set()


for prob, author1, author2, rec_type in primary_secondary_recommendations_list:
    if author_density_dict[author1] > 0 and author_density_dict[author2] > 0:
# i.e. 21-10-2020 neither is an isolated node (BUG NEEDS FIXING & INVESTIGATION)!
        if rec_type == "primary":
            auth_recommended_dict["primary"].add(author1)
            auth_recommended_dict["primary"].add(author2)
            auth_recommended_dict["primary_pairs"].add(tuple((author1, author2)))
            auth_recommended_dict["general_pairs"].add(tuple((prob,author1,
author2,rec_type)))

        elif rec_type == "secondary":
            auth_recommended_dict["secondary"].add(author1)
            auth_recommended_dict["secondary"].add(author2)
            auth_recommended_dict["secondary_pairs"].add(tuple((author1, author2)))
            auth_recommended_dict["general_pairs"].add(tuple((prob,author1,
author2,rec_type)))

#Creating Recommendations table for recommendation output:
file = open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/5-Link
Prediction(Recommendations)/1-Preparing multitype nodes and edge
dataset/Hetergenous_dataset_final.pickle",'rb')
unpickler = pickle.Unpickler(file)
Hetero_dataset = unpickler.load()
#Creating node density dictionaries to calculate standardised degree for each node- default
0 (no edges):
general_density_dict = dict()
for author in author_nodes.index:
    general_density_dict[author] = 0
for org in org_density_dict:
    general_density_dict[org] = 0
for venue in venue_nodes.index:
    general_density_dict[venue] = 0
for topic in topic_nodes.index:
    general_density_dict[topic] = 0
for source, target, relation, weight in Hetero_dataset.values:
    if source.replace(" ", "_") in general_density_dict:
        general_density_dict[source.replace(" ", "_")] += 1
    elif source.replace(" ", "_") in UNSW_dict:
        if UNSW_dict[source.replace(" ", "_")] in general_density_dict:
            general_density_dict[UNSW_dict[source.replace(" ", "_")]] += 1
        else:
            general_density_dict[UNSW_dict[source.replace(" ", "_")]] = 1
    elif source.replace(" ", "_") in Venue_clean_dict:
        if Venue_clean_dict[source.replace(" ", "_")] in general_density_dict:
            general_density_dict[Venue_clean_dict[source.replace(" ", "_")]] += 1
        else:
            general_density_dict[Venue_clean_dict[source.replace(" ", "_")]] = 1
    else:
        general_density_dict[source.replace(" ", "_")] = 0
    if target.replace(" ", "_") in general_density_dict:
        general_density_dict[target.replace(" ", "_")] += 1

    elif target.replace(" ", "_") in UNSW_dict:
        if UNSW_dict[target.replace(" ", "_")] in general_density_dict:
            general_density_dict[UNSW_dict[target.replace(" ", "_")]] += 1
        else:
```

```
            general_density_dict[UNSW_dict[target.replace(" ", "_")]] = 1

    elif target.replace(" ", "_") in Venue_clean_dict:
        if Venue_clean_dict[target.replace(" ", "_")] in general_density_dict:
            general_density_dict[Venue_clean_dict[target.replace(" ", "_")]] += 1
        else:
            general_density_dict[Venue_clean_dict[target.replace(" ", "_")]] = 1

    else:
        general_density_dict[target.replace(" ", "_")] = 0

author_coauthors = {}
author_orgs = {}
author_venues = {}
author_topics = {}
for _, prob,_, author1, author2 in Recommendations_dts.values:
    if (author1 in primary_authors_set or author2 in primary_authors_set) and (prob > 0)
and \
    (not tuple((author1,author2)) in coauthor_set) and (not tuple((author2,author1)) in
coauthor_set) and\
        (general_density_dict[author1] > 0 and general_density_dict[author2] > 0): #NOTE:
This is filtering out bug for
            # 0 edge nodes (isolated) to be included in recommendations (need to
investigate this to see
            #why these are being predicted for edges and if some nodes etc missing)

        print("Beginning",author1,author2)
        for source, target, relation, weight in Hetero_dataset.values:
            entity_org1 = None
            general_density_org1 = None
            entity_venue1 = None
            general_density_venue1 = None
            entity_topic1 = None
            general_density_topic1 = None
            entity_org2 = None
            general_density_org2 = None
            entity_venue2 = None
            general_density_venue2 = None
            entity_topic2 = None
            general_density_topic2 = None
            entity_coauthor1 = None
            entity_coauthor2 = None
            general_density_coauthor1 = None
            general_density_coauthor2 = None
            cnt_coauthor1 = None
            cnt_coauthor2 = None
            cnt_org1 = None
            cnt_venue1 = None
            cnt_topic1 = None
            cnt_org2 = None
            cnt_venue2 = None
            cnt_topic2 = None
            if (source == author1 or target == author1) and relation == 'AUTHOR-(P)-ORG':
                if source != author1:
                    if source.replace(" ", "_") in UNSW_dict:
                        entity_org1 = UNSW_dict[source.replace(" ", "_")]
                        general_density_org1 =
general_density_dict[UNSW_dict[source.replace(" ", "_")]]
                        cnt_org1 =  weight
                    elif source.replace(" ","_") in Venue_clean_dict:
                        entity_org1 = Venue_clean_dict[source.replace(" ", "_")]
```

```
                        general_density_org1 =
general_density_dict[Venue_clean_dict[source.replace(" ", "_")]]
                        cnt_org1 =  weight
                    elif not source.replace(" ", "_") in UNSW_dict and not source.replace("
", "_") in Venue_clean_dict:
                        entity_org1 = source.replace(" ", "_")
                        general_density_org1 = general_density_dict[source.replace(" ",
"_")]
                        cnt_org1 =  weight

                elif target != author1:
                    if target.replace(" ", "_") in UNSW_dict:
                        entity_org1 = UNSW_dict[target.replace(" ", "_")]
                        general_density_org1 =
general_density_dict[UNSW_dict[target.replace(" ", "_")]]
                        cnt_org1 =  weight
                    elif target.replace(" ","_") in Venue_clean_dict:
                        entity_org1 = Venue_clean_dict[target.replace(" ", "_")]
                        general_density_org1 =
general_density_dict[Venue_clean_dict[target.replace(" ", "_")]]
                        cnt_org1 =  weight
                    elif not target.replace(" ", "_") in UNSW_dict and not target.replace("
", "_") in Venue_clean_dict:
                        entity_org1 = target.replace(" ", "_")
                        general_density_org1 = general_density_dict[target.replace(" ",
"_")]
                        cnt_org1 =  weight
            if (source == author2 or target == author2) and relation == 'AUTHOR-(P)-ORG':
                if source != author2:
                    if source.replace(" ", "_") in UNSW_dict:
                        entity_org2 = UNSW_dict[source.replace(" ", "_")]
                        general_density_org2 =
general_density_dict[UNSW_dict[source.replace(" ", "_")]]
                        cnt_org2 = weight
                    elif source.replace(" ", "_") in Venue_clean_dict:
                        entity_org2 = Venue_clean_dict[source.replace(" ", "_")]
                        general_density_org2 =
general_density_dict[Venue_clean_dict[source.replace(" ", "_")]]
                        cnt_org2 = weight
                    elif not source.replace(" ", "_") in UNSW_dict and not source.replace("
","_") in Venue_clean_dict:
                        entity_org2 = source.replace(" ", "_")
                        general_density_org2 = general_density_dict[source.replace(" ",
"_")]
                        cnt_org2 = weight
                elif target != author2:
                    if target.replace(" ", "_") in UNSW_dict:
                        entity_org2 = UNSW_dict[target.replace(" ", "_")]
                        general_density_org2 =
general_density_dict[UNSW_dict[target.replace(" ", "_")]]
                        cnt_org2 = weight
                    elif target.replace(" ", "_") in Venue_clean_dict:
                        entity_org2 = Venue_clean_dict[target.replace(" ", "_")]
                        general_density_org2 =
general_density_dict[Venue_clean_dict[target.replace(" ", "_")]]
                        cnt_org2 = weight
                    elif not target.replace(" ", "_") in UNSW_dict and not target.replace("
","_") in Venue_clean_dict:
                        entity_org2 = target.replace(" ", "_")
                        general_density_org2 = general_density_dict[target.replace(" ",
"_")]
```

```
                    cnt_org2 = weight
            if (source == author1 or target == author1) and relation ==  'AUTHOR-(P)-VENUE'
:
                if source != author1:
                    entity_venue1 = source
                    general_density_venue1 = general_density_dict[source.replace(" ", "_")]
                    cnt_venue1 = weight
                elif target != author1:
                    entity_venue1 = target
                    general_density_venue1 = general_density_dict[target.replace(" ", "_")]
                    cnt_venue1 = weight
            if (source == author2 or target == author2) and relation ==  'AUTHOR-(P)-VENUE'
:
                if source != author2:
                    entity_venue2 = source
                    general_density_venue2 = general_density_dict[source.replace(" ", "_")]
                    cnt_venue2 = weight
                elif target != author2:
                    entity_venue2 = target
                    general_density_venue2 = general_density_dict[target.replace(" ", "_")]
                    cnt_venue2 = weight

            if (source == author1 or target == author1) and relation ==  'AUTHOR-(P)-TOPIC'
:
                if source != author1:
                    entity_topic1 = source
                    general_density_topic1 = general_density_dict[source.replace(" ", "_")]
                    cnt_topic1 = weight
                elif target != author1:
                    entity_topic1 = target
                    general_density_topic1 = general_density_dict[target.replace(" ", "_")]
                    cnt_topic1 = weight
            if (source == author2 or target == author2) and relation ==  'AUTHOR-(P)-TOPIC'
:
                if source != author2:
                    entity_topic2 = source
                    general_density_topic2 = general_density_dict[source.replace(" ", "_")]
                    cnt_topic2 = weight
                elif target != author2:
                    entity_topic2 = target
                    general_density_topic2 = general_density_dict[target.replace(" ", "_")]
                    cnt_topic2 = weight
            if (source == author1 or target == author1) and relation ==  'AUTHOR-(P)-
AUTHOR' :
                if source != author1:
                    entity_coauthor1 = source
                    general_density_coauthor1 = general_density_dict[source.replace(" ",
"_")]
                    cnt_topic1 = weight
                elif target != author1:
                    entity_coauthor1 = target
                    general_density_coauthor1 = general_density_dict[target.replace(" ",
"_")]
                    cnt_coauthor1 = weight
            if (source == author2 or target == author2) and relation ==  'AUTHOR-(P)-
AUTHOR' :
                if source != author2:
                    entity_coauthor2 = source
                    general_density_coauthor2 = general_density_dict[source.replace(" ",
"_")]
                    cnt_topic2 = weight
```

```
                elif target != author2:
                    entity_coauthor2 = target
                    general_density_coauthor2 = general_density_dict[target.replace(" ",
"_")]
                    cnt_coauthor2 = weight
            if (entity_coauthor1 != None):
                if author1 in author_coauthors:

author_coauthors[author1].append((entity_coauthor1,general_density_coauthor1,weight ))
                else:

author_coauthors[author1]=[(entity_coauthor1,general_density_coauthor1,cnt_coauthor1)]

            if (entity_org1!= None):
                if author1 in author_orgs:
                    author_orgs[author1].append((entity_org1,general_density_org1,weight ))
                else:
                    author_orgs[author1]=[(entity_org1,general_density_org1,cnt_org1)]
            if (entity_venue1!=  None):
                if author1 in author_venues:

author_venues[author1].append((entity_venue1,general_density_venue1,cnt_venue1 ))
                else:
                    author_venues[author1]=
[(entity_venue1,general_density_venue1,cnt_venue1)]

            if (entity_topic1 != None):
                if author1 in author_topics:

author_topics[author1].append((entity_topic1,general_density_topic1,cnt_topic1 ))
                else:
                    author_topics[author1] =
[(entity_topic1,general_density_topic1,cnt_topic1) ]
            if (entity_org2 !=None):
                if author2 in author_orgs:
                    author_orgs[author2].append((entity_org2,general_density_org2,cnt_org2
))
                else:
                    author_orgs[author2] = [(entity_org2,general_density_org2,cnt_org2)]
            if (entity_venue2 != None):
                if author2 in author_venues:

author_venues[author2].append((entity_venue2,general_density_venue2,cnt_venue2 ))
                else:
                    author_venues[author2] =
[(entity_venue2,general_density_venue2,cnt_venue2) ]
            if (entity_topic2 != None):
                if author2 in author_topics:

author_topics[author2].append((entity_topic2,general_density_topic2,cnt_topic2 ))
                else:
                    author_topics[author2] =
[(entity_topic2,general_density_topic2,cnt_topic2) ]
            if (entity_coauthor2 != None):
                if author2 in author_coauthors:

author_coauthors[author2].append((entity_coauthor2,general_density_coauthor2,weight ))
                else:

author_coauthors[author2]=[(entity_coauthor2,general_density_coauthor2,cnt_coauthor2)]
```

```
# author1   author2
#org   author1-degree   author2-degree   general-degree

Indirect_relationship_recommendation_pairs = set()
recommendation_intersection_pair_set = set()
recommendation_table = []
for _, prob,_, author1, author2 in Recommendations_dts.values:
    if (author1 in primary_authors_set or author2 in primary_authors_set) and (prob > 0)
and \
    (not tuple((author1,author2)) in coauthor_set) and (not tuple((author2,author1)) in
coauthor_set):
        if author1 in primary_authors_set and author2 in primary_authors_set:
            Recommendation_Type = "Primary Recommendation"
        elif (author1 in primary_authors_set or author2 in primary_authors_set) and not
(author1 in primary_authors_set and author2 in primary_authors_set):
            Recommendation_Type = "Secondary Recommendation"
        author1_topic_set = set()
        author2_topic_set = set()
        intersection = set()
        if author1 in author_topics and author2 in author_topics:
           for entity, general, specific in author_topics[author1]:
               author1_topic_set.add(entity)
           for entity, general, specific in author_topics[author2]:
               author2_topic_set.add(entity)
           intersection = author1_topic_set.intersection(author2_topic_set)
           print(intersection)
           if len(intersection) >= 1:
              for entity_int in intersection:
                   for entity, general, specific in author_topics[author1]:
                       if entity == entity_int:
                           auth1_spec = specific
                           general_int = general
                   for entity, general, specific in author_topics[author2]:
                       if entity == entity_int:
                           auth2_spec = specific
                           general_int = general
                   #if entity_int in topic_label_dict:
                   #    entity_int = topic_label_dict[topic]
                   if entity_int in Inferred_topic_label_dict:
                       entity_int = Inferred_topic_label_dict[entity_int]
                   recommendation_intersection_pair_set.add(tuple((author1, author2)))
                   Indirect_relationship_recommendation_pairs.add(tuple((author1,author2)))
                   recommendation_table.append([Recommendation_Type,author1,
author2,"Topic",entity_int,auth1_spec, auth2_spec,general_int,prob])
        author1_org_set = set()
        author2_org_set = set()
        intersection = set()
        if author1 in author_orgs and author2 in author_orgs:
           for entity, general, specific in author_orgs[author1]:
               author1_org_set.add(entity)
           for entity, general, specific in author_orgs[author2]:
               author2_org_set.add(entity)
           intersection = author1_org_set.intersection(author2_org_set)
           print(intersection)
           if len(intersection) >= 1:
              for entity_int in intersection:
                   for entity, general, specific in author_orgs[author1]:
                       if entity == entity_int:
                           auth1_spec = specific
                           general_int = general
                   for entity, general, specific in author_orgs[author2]:
```

```
                        if entity == entity_int:
                            auth2_spec = specific
                            general_int = general
                    recommendation_intersection_pair_set.add(tuple((author1, author2)))
                    Indirect_relationship_recommendation_pairs.add(tuple((author1,author2)))
                    recommendation_table.append([Recommendation_Type,author1,
author2,"Organisation",entity_int,auth1_spec, auth2_spec,general_int,prob])
        author1_venue_set = set()
        author2_venue_set = set()
        intersection = set()
        if author1 in author_venues and author2 in author_venues:
           for entity, general, specific in author_venues[author1]:
               author1_venue_set.add(entity)
           for entity, general, specific in author_venues[author2]:
               author2_venue_set.add(entity)
           intersection = author1_venue_set.intersection(author2_venue_set)
           print(intersection)
           if len(intersection) >= 1:
              for entity_int in intersection:
                    for entity, general, specific in author_venues[author1]:
                        if entity == entity_int:
                            auth1_spec = specific
                            general_int = general
                    for entity, general, specific in author_venues[author2]:
                        if entity == entity_int:
                            auth2_spec = specific
                            general_int = general
                    recommendation_intersection_pair_set.add(tuple((author1, author2)))
                    Indirect_relationship_recommendation_pairs.add(tuple((author1,author2)))
                    recommendation_table.append([Recommendation_Type,author1,
author2,"Venue",entity_int,auth1_spec, auth2_spec,general_int,prob])
        author1_coauthor_set = set()
        author2_coauthor_set = set()
        intersection = set()
        if author1 in author_coauthors and author2 in author_coauthors:
           for entity, general, specific in author_coauthors[author1]:
               author1_coauthor_set.add(entity)
           for entity, general, specific in author_coauthors[author2]:
               author2_coauthor_set.add(entity)
           intersection = author1_coauthor_set.intersection(author2_coauthor_set)
           print(intersection)
           if len(intersection) >= 1:
              for entity_int in intersection:
                    for entity, general, specific in author_coauthors[author1]:
                        if entity == entity_int:
                            auth1_spec = specific
                            general_int = general
                    for entity, general, specific in author_coauthors[author2]:
                        if entity == entity_int:
                            auth2_spec = specific
                            general_int = general
                    recommendation_intersection_pair_set.add(tuple((author1, author2)))
                    Indirect_relationship_recommendation_pairs.add(tuple((author1,author2)))
                    recommendation_table.append([Recommendation_Type,author1,
author2,"Coauthor",entity_int,auth1_spec, auth2_spec,general_int,prob])

#Checking to see if recommendation pairs are NOT in final recommendation summary due to
finding no DIRECT relationships shared between recommended researchers:
for prob, author1, author2, rec_type in auth_recommended_dict["general_pairs"]:
    if not tuple((author1,author2)) in recommendation_intersection_pair_set:
        if author1 in primary_authors_set and author2 in primary_authors_set:
```

```
            Recommendation_Type = "Primary Recommendation"
        elif (author1 in primary_authors_set or author2 in primary_authors_set) and not
(author1 in primary_authors_set and author2 in primary_authors_set):
            Recommendation_Type = "Secondary Recommendation"
        recommendation_table.append([Recommendation_Type,author1, author2,"No-Direct-
Entity-Intersection","Use Pair Indirect Co-authorship Relationship-Network Vizualisation
file","in", "'MuxViz Indirect-Network-Recommendation-Vizualisation' Folder","File:
{},{}".format(author1,author2),prob])
        Indirect_relationship_recommendation_pairs.add(tuple((author1,author2)))
#Converting and pickling recommendation summary dataset into a dataframe containing each
relation, edge set and weight instance between the networks.
recommendation_table_df = pd.DataFrame(recommendation_table, columns = ["Recommendation
Type","Researcher1", "Researcher2", "Intersection","Entity","Researcher1-Entity-
Weight","Researcher2-Entity-Weight","Entity-Network-Weight", "Research Collaboration
Probability"])
recommendation_table_df.to_csv("C:/Users/Liam
Ephraims/Desktop/LiamDissertation/Evaluation/5-Link
Prediction(Recommendations)/Recommendation_Summary.csv")

#Writing researcher pairs without direct relation intersections to name txt file for
creating later co-authorship vizualisations targeting these two authors only:
with open("C:/Users/Liam Ephraims/Desktop/LiamDissertation/Evaluation/5-Link
Prediction(Recommendations)/Indirect-Network-Recommendation-Vizualisations/Indirect-
relationship-recommendation-pairs.txt","w") as file:
    for author1, author2 in Indirect_relationship_recommendation_pairs:
        file.write("{}|{}\n".format(author1,author2))
file.close()


index_count = 0
index_dict = dict()
node_keep_dict = dict()
with open("./muxViz-master/MultiGraph_node_layout.txt", "w", encoding="utf-8") as file:
    file.write("nodeID nodeLabel\n")
    for author in author_density_dict:
        if author in Primary_author_dict:
            if (Primary_author_dict[author] == True and author in primary_authors_set):
               #   or (author in auth_recommended_dict["primary"]) or  (author in
auth_recommended_dict["secondary"]):
                index_count += 1
                index_dict[author] = index_count
                row_str = "{} {}\n".format(index_dict[author], author)
                print(row_str)
                node_keep_dict[author] = True
                file.write(row_str)
            else:
                node_keep_dict[author] = False
    for org in org_density_dict:
        if org in Primary_org_dict and not org in UNSW_dict:
            if (Primary_org_dict[org] == True and org_density_dict[org] >=
org_density_threshold): #or (org_density_dict[org] >= org_density_threshold):
                index_count += 1
                index_dict[org] = index_count
                row_str = "{} {}\n".format(index_dict[org], org)
                print(row_str)
                file.write(row_str)
                node_keep_dict[org] = True
            else:
                node_keep_dict[org] = False
        elif org in Primary_org_dict and org in UNSW_dict:
```

```
            if (Primary_org_dict[UNSW_dict[org]] == True and
org_density_dict[UNSW_dict[org]] >= org_density_threshold): #or (org_density_dict[org] >=
org_density_threshold):
                if UNSW_dict[org] in index_dict:
                    continue
                    #want a unique id for just the one primary org name - not each error
org name
                if not UNSW_dict[org] in index_dict:
                    index_count += 1
                    index_dict[UNSW_dict[org]] = index_count
                row_str = "{} {}\n".format(index_dict[UNSW_dict[org]], UNSW_dict[org])
                print(row_str)
                file.write(row_str)
                node_keep_dict[UNSW_dict[org]] = True
            #else:
    for venue in venue_density_dict:
        if venue in Primary_venue_dict and not venue in Venue_clean_dict:
            if (Primary_venue_dict[venue] == True and venue_density_dict[venue] >=
venue_density_threshold): #or (venue_density_dict[venue] >= venue_density_threshold):
                index_count += 1
                index_dict[venue] = index_count
                row_str = "{} {}\n".format(index_dict[venue], venue)
                print(row_str)
                file.write(row_str)
                node_keep_dict[venue] = True
            else:
                node_keep_dict[venue] = False
        elif venue in Primary_venue_dict and venue in Venue_clean_dict:
            if (Primary_venue_dict[Venue_clean_dict[venue]] == True and
venue_density_dict[Venue_clean_dict[venue]] >= venue_density_threshold): #or
(org_density_dict[org] >= org_density_threshold):
                if Venue_clean_dict[venue] in index_dict:
                    continue
                    #want a unique id for just the one correct name - not each error venue
name
                if not Venue_clean_dict[venue] in index_dict:
                    index_count += 1
                    index_dict[Venue_clean_dict[venue]] = index_count
                row_str = "{} {}\n".format(index_dict[Venue_clean_dict[venue]],
Venue_clean_dict[venue])
                print(row_str)
                file.write(row_str)
                node_keep_dict[Venue_clean_dict[venue]] = True
            else:
                node_keep_dict[Venue_clean_dict[venue]] = False
    for topic in topic_density_dict:
        if topic in Primary_topic_dict:
            if (Primary_topic_dict[topic] == True and topic_density_dict[topic] >=
topic_density_threshold): #or (topic_density_dict[topic] >= topic_density_threshold):
                index_count += 1
                #if topic in Inferred_topic_label_dict:
                topic_label = Inferred_topic_label_dict[topic]
                topic_label = topic_label.replace(" ", "_")
                index_dict[topic] = index_count
                row_str = "{} {}\n".format(index_dict[topic], topic_label)
                print(row_str)
                file.write(row_str)
                node_keep_dict[topic] = True
            else:
                node_keep_dict[topic] = False
```

```
    file.close()
print(index_count)

#Creating node colour and size file - NEW version only primary nodes and non-primary nodes
above threshold for degree:
cnt = 0
recom_author_cnt = 0
with open("./muxViz-master/MultiGraph_node_colour_size.txt", "w", encoding="utf-8") as
file:
    file.write("nodeID layerID color size\n")
    for node in node_keep_dict:
        #Primary node: ORANGE
        if node_keep_dict[node] == True and node in Primary_org_dict:
                    row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Organisations"], "green",org_density_dict[node]*100)
                    print(row_str)
                    cnt += 1
                    file.write(row_str)
        elif node_keep_dict[node] == True and node in Primary_author_dict:
            if Primary_author_dict[node] == True:
                if (node in primary_authors_set) and not ((node in
auth_recommended_dict["primary"]) or  (node in auth_recommended_dict["secondary"])) :
                    row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Authors"], "orange",author_density_dict[node]*20)
                    print(row_str)
                    cnt += 1
                    file.write(row_str)
                elif (node in primary_authors_set) and ((node in
auth_recommended_dict["primary"]) or  (node in auth_recommended_dict["secondary"])) :
                    print("Recommended author node is", node, "as primary node will not
amend node colour or size!")
                    recom_author_cnt += 1
                    row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Authors"], "purple",author_density_dict[node]*20)
                    print(row_str)
                    cnt += 1
                    file.write(row_str)
               #elif (not node in primary_authors_set) and ((node in
auth_recommended_dict["primary"]) or  (node in auth_recommended_dict["secondary"])) :
               #     print("Recommended author node is", node, "and not primary node will
amend colour to red!")
               #     recom_author_cnt += 1
               #     row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Authors"], "red",author_density_dict[node]*100)
        #!HERE AMENDED WILL NEED FIXING!!!!!!!!!!!!!!!! for recommended authors!

                    #row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Authors"], "purple",1)

        elif node in Primary_venue_dict and node_keep_dict[node] == True:
                    row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Venues(Journal/Conferences)"], "blue",venue_density_dict[node]*20)
                    print(row_str)
                    cnt += 1
                    file.write(row_str)
        elif node in Primary_topic_dict and node_keep_dict[node] == True:
                    row_str = "{} {} {} {}\n".format(index_dict[node],
layer_dict["Topics"], "red",topic_density_dict[node]*20)
                    print(row_str)
                    cnt += 1
                    file.write(row_str)
```

```
included_relations_set = set()
Hetero_primary_flag_dts_pruned = []
for source, target, relation, weight, primary_flag in Hetero_primary_flag_dts:
   if primary_flag == True:
       Hetero_primary_flag_dts_pruned.append([source, target, relation, weight,
primary_flag])
   if primary_flag == False:
        #note: non-primary edge weight <= 50th percentile (half of edges) already removed.
        #Make sure source or target are not excluded nodes:
        if source.replace(" ", "_") in node_keep_dict and target.replace(" ", "_") in
node_keep_dict:
            included_relations_set.add(relation)
            if node_keep_dict[source.replace(" ", "_")] == True and
node_keep_dict[target.replace(" ", "_")] == True:
                Hetero_primary_flag_dts_pruned.append([source, target, relation, weight,
primary_flag])
        #Otherwise, if not a primary author edge is the edge a primary organisation (UNSW)
edge?
        elif source.replace(" ", "_") in UNSW_dict or target.replace(" ", "_") in
UNSW_dict:
            primary_org = False
            if source.replace(" ", "_")  in UNSW_dict:
                if  UNSW_dict[source.replace(" ", "_")] == Correct_primary_org_name:
                    primary_org = True
            elif target.replace(" ", "_")  in UNSW_dict:
                if  UNSW_dict[target.replace(" ", "_")] == Correct_primary_org_name:
                    primary_org = True
            if primary_org == True:
                Hetero_primary_flag_dts_pruned.append([source, target, relation, weight,
primary_flag])

        #Looking for primary author edge instances:
        if  ((source in primary_authors_set) or (target in primary_authors_set)): #and
((not source in auth_recommended_dict["primary"]) and (not target in
auth_recommended_dict["secondary"])):
            if source.replace(" ", "_") in org_nodes.index:
                layerID_from = layer_dict["Organisations"]

            elif source.replace(" ", "_") in venue_nodes.index:
                layerID_from = layer_dict["Venues(Journal/Conferences)"]
            elif source.replace(" ", "_") in topic_nodes.index:
                layerID_from = layer_dict["Topics"]
            elif source.replace(" ", "_") in author_nodes.index:
                layerID_from = layer_dict["Authors"]
            else:
                print("EXCEPTION!!!", source)
            if target.replace(" ", "_") in org_nodes.index:
                layerID_to = layer_dict["Organisations"]
            elif target.replace(" ", "_") in venue_nodes.index:
                layerID_to = layer_dict["Venues(Journal/Conferences)"]
            elif target.replace(" ", "_") in topic_nodes.index:
                layerID_to = layer_dict["Topics"]
            elif target.replace(" ", "_") in author_nodes.index:
                layerID_to = layer_dict["Authors"]
            else:
                print("EXCEPTION!!!", target)

            if source.replace(" ", "_") in UNSW_dict:
                source = UNSW_dict[source.replace(" ", "_")]
            else:
```

```
                source = source.replace(" ", "_")
            if target.replace(" ", "_") in UNSW_dict:
                target = UNSW_dict[target.replace(" ", "_")]
            else:
                target = target.replace(" ", "_")

            if source.replace(" ", "_") in Venue_clean_dict:
                source = Venue_clean_dict[source.replace(" ", "_")]
            else:
                source = source.replace(" ", "_")
            if target.replace(" ", "_") in Venue_clean_dict:
                target = Venue_clean_dict[target.replace(" ", "_")]
            else:
                target = target.replace(" ", "_")
            #Giving emphasis to primary author edges by increasing weight of primary author
edges by multiple of 4.
            if source in index_dict and target in index_dict:
                row_str = "{} {} {} {} {} {}\n".format(index_dict[source],layerID_from,
                                                       index_dict[target], layerID_to,
                                                       "orange", round(weight*4,4))


                file.write(row_str)
                cnt += 1
        #Creating green edges for primary organisation connections.
        # No extra emphasis is given to weights of primary organisation (UNSW):
        #Now looking for secondary case, primary organisation edge instances:
        elif  ((source.replace(" ", "_") in UNSW_dict or target.replace(" ", "_") in
UNSW_dict)):
            primary_org_edge = False
            #print(source, target, "PRIMARY ORG EDGE")
            if source.replace(" ", "_") in UNSW_dict:
                if UNSW_dict[source.replace(" ", "_")] == Correct_primary_org_name:
                    primary_org_edge = True
            if target.replace(" ", "_") in UNSW_dict:
                if UNSW_dict[target.replace(" ", "_")] == Correct_primary_org_name:
                    primary_org_edge = True
            if primary_org_edge == True:
                if source.replace(" ", "_") in org_nodes.index:
                    layerID_from = layer_dict["Organisations"]

                elif source.replace(" ", "_") in venue_nodes.index:
                    layerID_from = layer_dict["Venues(Journal/Conferences)"]
                elif source.replace(" ", "_") in topic_nodes.index:
                    layerID_from = layer_dict["Topics"]
                elif source.replace(" ", "_") in author_nodes.index:
                    layerID_from = layer_dict["Authors"]
                else:
                    print("EXCEPTION!!!", source)
                if target.replace(" ", "_") in org_nodes.index:
                    layerID_to = layer_dict["Organisations"]
                elif target.replace(" ", "_") in venue_nodes.index:
                    layerID_to = layer_dict["Venues(Journal/Conferences)"]
                elif target.replace(" ", "_") in topic_nodes.index:
                    layerID_to = layer_dict["Topics"]
                elif target.replace(" ", "_") in author_nodes.index:
                    layerID_to = layer_dict["Authors"]
                else:
                    print("EXCEPTION!!!", target)

                if source.replace(" ", "_") in UNSW_dict:
```

```
                    source = UNSW_dict[source.replace(" ", "_")]
                else:
                    source = source.replace(" ", "_")
                if target.replace(" ", "_") in UNSW_dict:
                    target = UNSW_dict[target.replace(" ", "_")]
                else:
                    target = target.replace(" ", "_")

                if source.replace(" ", "_") in Venue_clean_dict:
                    source = Venue_clean_dict[source.replace(" ", "_")]
                else:
                    source = source.replace(" ", "_")
                if target.replace(" ", "_") in Venue_clean_dict:
                    target = Venue_clean_dict[target.replace(" ", "_")]
                else:
                    target = target.replace(" ", "_")

                if source in index_dict and target in index_dict:
                    row_str = "{} {} {} {} {} {}\n".format(index_dict[source],layerID_from,
                                                           index_dict[target], layerID_to,
                                                           "green", round(weight*2,4))


                    file.write(row_str)
                    cnt += 1
    file.close()
print("{} edges".format(cnt))

 #Extended edge list format - NEW ONLY PRIMARY AUTHOR AND edges with >= 10 EDGES and from
non-excluded nodes:
#[source node] [source layer][target[target layer] [weight]
#for relation in relations:
edge_cnt = 0
file_name = "./muxViz-master/MultiGraph_Layer_edges_extended.edges"
with open(file_name, "w", encoding="utf-8") as file:
    for source, target, relation, weight, primary_flag in Hetero_primary_flag_dts_pruned:
        if source.replace(" ", "_") in venue_nodes.index:
            slayer = "Venues(Journal/Conferences)"
        elif source.replace(" ", "_")  in org_nodes.index:
            slayer = "Organisations"
        elif source.replace(" ", "_")  in author_nodes.index:
            slayer = "Authors"
        elif source.replace(" ", "_")  in topic_nodes.index:
            slayer="Topics"
        if target.replace(" ", "_") in venue_nodes.index:
            tlayer = "Venues(Journal/Conferences)"
        elif target.replace(" ", "_") in org_nodes.index:
            tlayer = "Organisations"
        elif target.replace(" ", "_") in author_nodes.index:
            tlayer = "Authors"
        elif target.replace(" ", "_") in topic_nodes.index:
            tlayer="Topics"
        edge_cnt += 1

        if source.replace(" ", "_") in UNSW_dict:
            source = UNSW_dict[source.replace(" ", "_")]
        else:
            source = source.replace(" ", "_")
        if target.replace(" ", "_") in UNSW_dict:
            target = UNSW_dict[target.replace(" ", "_")]
        else:
```

```
            target = target.replace(" ", "_")

        if source.replace(" ", "_") in Venue_clean_dict:
            source = Venue_clean_dict[source.replace(" ", "_")]
        else:
            source = source.replace(" ", "_")
        if target.replace(" ", "_") in Venue_clean_dict:
            target = Venue_clean_dict[target.replace(" ", "_")]
        else:
            target = target.replace(" ", "_")

        if source in index_dict and target in index_dict:
            row_str = "{} {} {} {} {}\n".format(index_dict[source],layer_dict[slayer],
                                               index_dict[target], layer_dict[tlayer],
                                               round(weight, 4))
            file.write(row_str)
        else:
            print(source,target,"not in index dict" )
    file.close()
print(edge_cnt)
#Configuration file to be imported to MuxViz as assembly file.
with open("./muxViz-master/MultiGraph_node_CONFIG.txt", "w", encoding="utf-8") as file:
    edge_file = "MultiGraph_Layer_edges_extended.edges"
    layer_file = "MultiGraph_Layers.txt"
    node_layout = "MultiGraph_node_layout.txt"
    row_str = "{};{};{}\n".format(edge_file,layer_file, node_layout)
    file.write(row_str)
    file.close()
```