




University  
of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

## MASTER'S THESIS

Study program/specialization: Computer Science	Spring semester, 2018  Open / <del>Confidential</del>
Author: Junaid Alam	 ..... (signature of author)
Programme coordinator:  Supervisor(s): Vinay Jayarama Setty (UiS), Derek Göbel, Paolo Predonzani (LOOPS)	
Title of Master's Thesis: <b>Graph-based Entity Recognition &amp; Inference and Link Prediction in static Network</b>	
Credits: 30 ECTS	
Keywords: Graph entity recognition, Inference, Link prediction	Number of pages: + supplemental material/other: - 90  Stavanger, June 15, 2018





Faculty of Science and Technology  
Department of Electrical Engineering and Computer Science

# Graph-based Entity Recognition & Inference and Link Prediction in static Network

Master's Thesis in Computer Science  
by

Junaid Alam

Internal Supervisors

Vinay Jayarama Setty

External Supervisors

Derek Göbel

Paolo Predonzani

June 14, 2018



# Declaration of Authorship

I, Junaid Alam, declare that this thesis titled, ‘Graph-based Entity Recognition & Inference and Link Prediction in static Network’ and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master’s degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

---

Date:

---



## DEDICATION

*“I can't change the direction of the wind, but I can adjust my sails to always reach my destination”*

Jimmy Dean

This thesis is dedicated to ALL Rohingyas in Myanmar, who are deprived of the right to free movement and the right to higher education.

## *Abstract*

The size of data we are producing is exponentially increasing every year. According to former Google CEO Eric Schmidt, we produce as much information in two days now as we did from the dawn of mankind through 2003. The Oil & Gas industries produce millions of linked data each day. However, a vast majority of the data are unstructured or semi-structured data. To make a good decision, it is very important that we know our data. Many industries rely on the insights of their data to take any further action. Therefore, it is very important for the advancement of a company or an institution to have an overall view of the data they are producing.

For this thesis, we studied some data produced by Oil & Gas industries that are provided to us by **LOOPS**, and we found that the data are usually linked data. Two linked data can be interlinked with each other and become more useful through semantic queries. However, due to poor presentation of the data, the benefit that can be achieved from linked data is lacking.

In this thesis, we devised a system that extracts the meaningful information from the semi-structured data and visualizes the data using the power of graph. We then use the graph to have the insights of the data. The system can recognize entities in the graph and give important feedbacks by inferring more knowledge about the recognized entities.

As we said, the data are interlinked with other data. However, usually in linked data, some of the links between the data might be missing. The more the data are linked, the more useful information we can learn from it. Therefore, we invested a significant portion of our research in predicting the possible missing links between data using supervised and unsupervised link prediction approach.





## *Acknowledgements*

I am very grateful to Derek Göbel and Paolo Predonzani, my external supervisors from **LOOPS** for their contentious support, instruction, and encouragement. Derek and Paolo have dedicated a significant portion of their valuable time to support me in this project. I can confidently say that if it was not Paolo who always guided me and pushed me to do a proper research, this work would have never been completed.

I would graciously take this opportunity to thank Vinay Jayarama Setty, my internal supervisor from **UiS** for his valuable advice and encouragement. Every time I meet him, he surprises me with his ingenious ideas and the way of his explanation.

Finally, I would like to thank my family, especially my elder brother Sayed Alam for his uninterrupted love and support. I would not be who I am today if it was not for his sacrifice.

# Contents

<b>Declaration of Authorship</b>	<b>iii</b>
<b>DEDICATION</b>	<b>v</b>
<b>Abstract</b>	<b>vi</b>
<b>Acknowledgements</b>	<b>viii</b>
<b>Abbreviations</b>	<b>xiii</b>
<b>Symbols</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Challenges . . . . .	2
1.3 Contributions . . . . .	2
1.4 Used Software . . . . .	3
1.5 Outline . . . . .	3
<b>2 Literature Review</b>	<b>5</b>
2.1 Semantic Web . . . . .	5
2.1.1 RDF- Resource Description Framework . . . . .	5
2.1.2 RDF-Schema . . . . .	6
2.1.3 Inference in Semantic Web . . . . .	7
2.2 Graph Database . . . . .	8
2.2.1 LPG - Label Property Graph . . . . .	10
2.2.2 Index Free Adjacency . . . . .	10
2.2.3 Graph Traversal . . . . .	10
2.2.4 Graph Query . . . . .	11
<b>3 Extracting information from Excel file for building graph</b>	<b>13</b>
3.1 Structure of the files . . . . .	13
3.1.1 Structure of Type I sheet . . . . .	14

3.1.2	Structure of TYPE II Sheet . . . . .	15
3.2	Extracting the Objects . . . . .	15
3.2.1	Data Normalization . . . . .	17
3.3	Extracting the Relationships . . . . .	18
3.3.1	Data Clean up . . . . .	20
3.4	Saving Objects and Relationships for Future use . . . . .	21
3.5	Experimental Setup and Result . . . . .	21
3.6	Conclusion . . . . .	23
<b>4</b>	<b>Implementation of Graph for Entity Recognition and Inference</b>	<b>25</b>
4.1	Introduction . . . . .	25
4.2	Type of Graphs . . . . .	25
4.3	Representation of graphs . . . . .	26
4.4	Representing LOOPS data in graph . . . . .	26
4.5	Graph-based Entity Recognition . . . . .	29
4.6	Graph-based Inference . . . . .	31
4.7	Experimental Setup and Result . . . . .	35
4.8	Conclusion . . . . .	35
<b>5</b>	<b>Link Prediction - Unsupervised Approach</b>	<b>37</b>
5.1	Introduction . . . . .	37
5.2	Graph notation and terms . . . . .	39
5.3	Data Preparation . . . . .	39
5.4	Experimental Setup and Evaluation Metric . . . . .	40
5.5	Link Prediction Methods . . . . .	42
5.5.1	Common Neighbors . . . . .	43
5.5.2	Jaccard's Coefficient . . . . .	43
5.5.3	Adamic/Adar Index . . . . .	43
5.5.4	Preferential Attachment . . . . .	44
5.5.5	Resource Allocation . . . . .	44
5.6	Result and Discussion . . . . .	44
<b>6</b>	<b>Link Prediction - Supervised Approach</b>	<b>47</b>
6.1	Supervised Learning . . . . .	47
6.2	Limitations of Unsupervised Link Prediction . . . . .	48
6.3	Dataset and Evaluation metrics . . . . .	48
6.4	Feature Selection . . . . .	49
6.5	Class imbalance and under-sampling . . . . .	50
6.6	Link Prediction using Supervised Learning . . . . .	51
6.7	Classification Algorithms . . . . .	53
6.8	Result and Discussion . . . . .	53
<b>7</b>	<b>Conclusion and Further Work</b>	<b>55</b>
7.1	Conclusion . . . . .	55
7.2	Further Work . . . . .	56

---

<b>List of Figures</b>	<b>57</b>
<b>List of Tables</b>	<b>61</b>
<b>A Appendix</b>	<b>65</b>
<b>Bibliography</b>	<b>71</b>



# Abbreviations

<b>LPG</b>	<b>L</b> abel <b>P</b> roperty <b>G</b> raph
<b>WWW</b>	<b>W</b> orld <b>W</b> ide <b>W</b> eb
<b>W3C</b>	<b>W</b> orld <b>W</b> ide <b>W</b> eb <b>C</b> onsortium
<b>RDF</b>	<b>R</b> esource <b>D</b> escription <b>F</b> ramework
<b>OWL</b>	<b>W</b> eb <b>O</b> ntology <b>L</b> anguage





# Symbols

symbol	description
$V$	set of Nodes in a graph
$E$	set of Edges in a graph
$G = \langle V, E \rangle$	a network G
$U$	universal set of edges in a graph
$X$	set of nonexistent edges
$ET$	set of edges of train set
$EP$	set of edges of test set
$\Gamma(x)$	neighbors of node x



# Chapter 1

## Introduction

### 1.1 Motivation

The size of data we are producing is exponentially increasing every year. According to former Google CEO Eric Schmidt, we produce as much information in two days now as we did from the dawn of mankind through 2003 [1]. Not all of the data we produced are structured data, many of them have no defined structures. How to turn the data we produced to set of meaningful information is a hot topic of research among Big Data community.

Business such as finance and oil sectors has always wanted to derive the insights from information to make a fact-based decision in a more smarter and real-time manner. At Loops, we analyze the data produced by oil and gas sectors and found that the data they produce are usually linked data that are interconnected with other data.

Getting information from the data manually is a tedious task and need a great demand of human resources and at some time become unfeasible. However, if the data are cleverly represented, we can not only extract the information from the big volume of data but also can infer other information that is hidden inside the data that are not easily detectable to human eyes. For example, in a spreadsheet produced by an employee at an oil company, one of the cells may have a string of text such as *Drilling at F-East*. From this piece of information, we understand nothing. However, if we can recognize entities such as F-East in the text, then can an answer a lot of questions such as what F-East is, where is it etc.

With these ideas in mind, we propose a system that represents the data in a directed graph and can be used to query various questions. Our main focus is on recognizing the entities from an input data set and inferring further knowledge that is not easily detectable to human eyes. We are also inspired to do an experiment on our graph to

predict possible missing links. If we can predict the missing link in a network, we can make the graph more complete by adding those potential missing links to the graph.

## 1.2 Challenges

Nodes and relationships are the building blocks of a graph, so we need to come up with an efficient idea of extracting nodes and relationships from the data available at our disposal and build a graph so that the system can use it to recognize and infer entities for an unseen dataset. The dataset that is provided by LOOPS contains so many redundant, conflicting and abnormal data. Hence, the first and the most important part of the work is to extract the information and clean them as much as possible. The presence of noisy data in a graph can contaminate the end result.

The inference problem we dealt in this thesis is not a rule-based inference but hierarchical inference. In hierarchical inference task, inferring knowledge is comparatively easy if two nodes are directly connected. However, when two nodes are connected through a variable length of hops, it is challenging to infer new knowledge.

In our dataset, we do not have enough features for a node. The nodes in the graph have only a name and a type attributes. Therefore, we have to use a relatively poor set of features for supervised link prediction. All the features we used are topological features such as *Common Neighbors*, *Jaccard Coefficient* etc.

## 1.3 Contributions

For this thesis, we make the following contributions:

- We develop *object extractor* - a system that effectively extracts all relevant objects and their relationships from Excel files. The *object extractor* eliminates data inconsistency and ambiguity. It also performs data normalization.
- We developed *ger*- a system that represents the data extracted by *object extractor* in a graph and recognizes the entities in an input data file using the graph. It also infers more knowledge for the recognized entity by using hierarchical inference techniques.
- We experiment supervised and unsupervised link prediction methods on the graph to predict possible missing links in the network.

## 1.4 Used Software

All the softwares we use in this thesis are freely available open-source softwares. For Entity recognition and Graph Inference, we use Java and for link prediction, we use Python.

### Software Used for Writing the Report:

1. LaTeX
2. TexMaker

### Programming Languages and Tools:

1. Java 1.8
2. JetBrains IntelliJ IDE (Student License)
3. Python 3.6
4. JetBrains PyCharm IDE(Student License)

### Libraries used:

1. **Apache POI** , a Java library for reading and writing files in MS Office formats
2. **Guava**, a common utility library developed by Google
3. **Pandas**, a Python library for data manipulation and analysis
4. **NetworkX**, a Python library for analyzing graphs and networks
5. **Scikit-learn**, a machine learning library for Python

## 1.5 Outline

The remainder of the thesis is structured as follows:

**Chapter 2** introduces literature review. In this chapter, we discuss Semantic Web and Graph Database. Understanding how Semantic Web and Graph Database represent linked data is an important part for this thesis.

**Chapter 3** presents the implementation of *object extractor* system that extracts the relevant information from Excel files provided by LOOPS. Here, our main focus is to extract only relevant information by successfully tackling data redundancy, data inconsistency and ambiguity that exist in raw dataset. The system normalizes the extracted data so that different representation of same data in different files are unified into a consistent source of information.

**Chapter 4** describes the implementation of *ger system* that represents information extracted by *object extractor* in a graph data model. In this chapter, we analyze different graphs and graph data representation techniques. We also discuss different graph traversal techniques and apply them in *ger system* for recognizing entities and inferring more knowledge about the recognized entities.

**Chapter 5** provides the missing link prediction methods using unsupervised approach. In this chapter, we experiment the network that we build using *ger system* to predict potential missing link in the network. We use different link prediction algorithms that use features intrinsic to the network topology such as *Common Neighbors*, *Jaccard Coefficient* etc.

**Chapter 6** uses different supervised machine learning algorithms for link prediction for the network we build using *ger system*. The unsupervised link prediction algorithm we implemented in Chapter 5 uses a single feature for predicting the missing links. Using a single feature may not completely explore different structural patterns contained in the network. Therefore, we use supervised machine learning algorithm that composes all the features into a set of feature vector and predict missing link.

**Chapter 7** concludes and presents suggestions for further work.

Since we develop different systems to achieve different goals for this thesis, instead of dedicating a separate chapter for the result and discussion, we opted to include the experimental setup, result and discussion for each system in their respective chapter. In doing so, readers can read each chapter independently and have a clear image of the system that we proposed.

## Chapter 2

# Literature Review

The literature review is an important part of any research. It helps the researcher get familiar with the domain of the research area and existing technology stacks. To understand the domain of our research, we study semantic web and graph database. Semantic web technology broaden our knowledge about the linked data and inference while graph database helps us understand many different techniques for graph analysis.

### 2.1 Semantic Web

The initial concept of World Wide Web was to put information to a computer by anyone and access that information by anyone anywhere. At the initial step, only people were able to discover the meaning of data on the web. However, the creator of WWW- Sir Tim Berners-Lee believes that eventually, machines would also be able to use the information on the web that would ultimately allow powerful and effective human-computer-human collaboration [2]. As a result, Semantic Web was born. Semantic Web is more a vision than a technology where the data located anywhere on the web is accessible and understandable to both human and machines. To bring the life to the vision of Semantic Web, different technologies have evolved such as *RDF*, *OWL*, *SPARQL* etc.

#### 2.1.1 RDF- Resource Description Framework

A huge amount of data available on the web are not uniformly formatted. To process the data by both machine and people, a standard format is needed. RDF is such a simple data model standardized by *W3C* for describing and modeling the knowledge and exchanging the information. In RDF, basically there are resources and there are statements that can be made about those resources. A statement links two resources

and usually has the form of *subject-predicate-object* where subject and object are two resources linked together by a predicate. Since an RDF statement always consists of three components, it is also known as RDF triple. An example of an RDF statement is

*Mark Zuckerberg is the owner of Facebook*

If we interpret this into a triple, we will have

(Mark Zuckerberg, isOwnerOf, Facebook)

We can think of the triple  $(x, P, y)$  as  $P(x,y)$  where predicate  $P$  relates the resource  $x$  to the resource  $y$ . RDF statements can be represented with a directed labeled graph where resources  $x$  and  $y$  of the statement are two nodes and the predicate  $P$  is the relationship or edge that connects  $x$  to  $y$ . A more specific definition of RDF graph is given at [3]:

**Definition 2.1.** An RDF graph is a directed labeled graph, denoted as  $G = (V, E, L_E)$ , where  $V$  is a set of nodes corresponding to subjects and objects and  $E$  is a set of directed edges from the subject to the objects.  $L_E$  is a set of edge labels referring to the predicates associated with the edges. A node with in-degree as zero is called as a source node.

For uniquely identifying resources in RDF statement, *Uniform Resource Identifiers (URIs)* are used as the label of the resources. For serialization of RDF, there exist different alternatives such as *RDF/XML*, *Turtle*, *N-Triple* etc. RDF triples are stored in RDF triplestore and are queried using SPARQL query language.

### 2.1.2 RDF-Schema

RDF is a universal language where a user defines the resources using their own vocabularies. To make the RDF domain specific, domain-specific vocabularies are used. These vocabularies are defined in RDF Schema. The *Mark Zuckerberg* and *Facebook* resources in above statement are individual objects. We also want to have Person, Website, Courses etc that define the individual object. For example, we can classify Mark Zuckerberg object as an instance of Person and Facebook as an instance of Website classes. Also, by using classes, we can put a restriction on what can be stated. For example, although they are valid RDF statements, we do not want to allow the statement such as:

*Mathematics is taught by Physics*

*Room 505-E is taught by John*

We do not want to allow the first statement because we want a course to be taught by a Professor only. To do that, we have to put a restriction on value of *taught by* property



i.e restricting the *range*. For the second statement, we need to put restriction so that only courses be taught. Hence we want to put a restriction on the subject resource on which we can apply property i.e restricting the *domain*.

RDF Schema allows hierarchies of classes. For example, every professor is an academic staff. We can say that professor is "a subclass of" academic staff. If class A is a subclass of B, then every instance of A is also an instance of B. Similar to class hierarchies, property hierarchies can be accomplished. RDF Schema is written in RDF triple format, hence every RDF Schema is a valid RDF document.

### 2.1.3 Inference in Semantic Web

Inference in semantic web mean discovering new statements or relationships between resources. Using the class hierarchies of RDF Schema, we can infer new relationships. For example, consider the following RDF in a RDF store:

```
:AcademicStaff    rdf:type      rdfs:Class
:Professor        rdfs:subClassOf  :AcademicStaff
:John            rdf:type      Professor
```

From the above, RDF triples, since **John** is a Professor and Professor is a subclass of Academic Stuff, we can **infer** that *John* is an Academic Stuff even though that statement is not explicitly included in original triples.

By using the hierarchies of the properties in RDF Schema, we can discover new knowledge. Consider the following triples:

```
:AcademicStaff    rdf:type      rdfs:Class
:Professor        rdfs:subClassOf  :AcademicStaff
:Course          rdf:type      rdfs:Class
:teaches         rdf:type      rdf:Property
:teaches         rdfs:domain     :Professor
:teaches         rdfs:range     :Course
:John            :teaches      :Mathematics
```

From the above triple, we can infer than **John** is a Professor and **Mathematics** is a Course even though the knowledge is not in original RDF triples. This is because, according to the RDF schema we defined, the domain of the *teaches* is a Professor and the range is a Course.

RDF Schema only allows limited inference such as subclass hierarchical inference. Sometimes, more complex logical reasoning is required to infer new knowledge. For, example,

the equivalence of classes, the intersection of classes etc cannot be defined in RDF Schema. For more complex reasoning, OWL-Web Ontology Language is a W3C recommendation. Explanation of how OWL works in Semantic Web is out of the scope of this research. Interested readers are encouraged to read [4].

## 2.2 Graph Database

Historically, relational databases are being used for storing, querying, retrieving and manipulating data. However, after the advent of the internet, data size is growing exponentially and for the highly connected linked data where there are a lot of many-to-many relationships, the relational model becomes a burden with large join tables and sparsely populated rows and a lot of null checking logic [5]. Since relational databases use a fixed schema which is not designed for frequent changes, dealing with the request of changes is a challenge and need considerable human input. As a result, graph database technologies evolve for efficiently handling connected data with the dynamic schema.

A graph database is a database management system with CRUD functionality like a relational database. Not all graph database use native graph storage, some store nodes and edges in a relational database. In this thesis, we will focus on native graph databases. Native graph storage stores information using graph data model.

Relational databases do not have the fixed relationship between records. Instead, the relationship between records exists at modeling time as a mean of join tables and foreign keys. Retrieving related data using JOIN operation is computationally costly. Since JOIN operations are so common in relational databases, relational databases are optimized for single JOIN. However, as the level of depth increases, the performance degrade exponentially. The performance degrades is because of the Cartesian product used by JOIN operation.

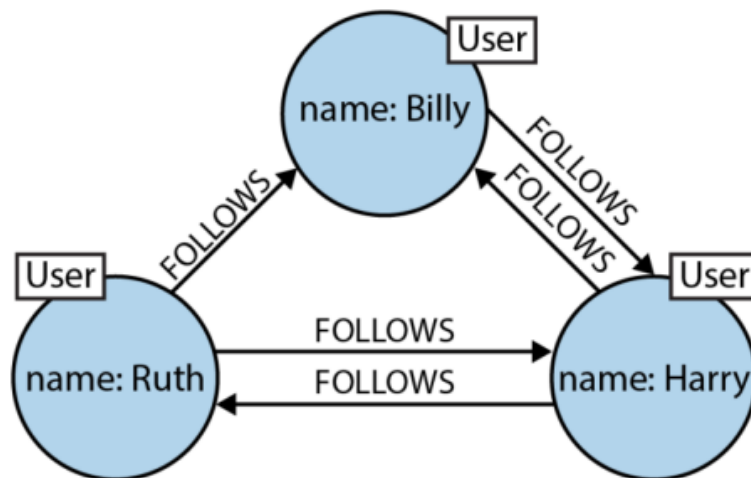
In contrast, the relationship is a first class citizen in graph databases hence there needs no JOIN operation. To get the related data, graph database uses graph traversal that navigates to its connected nodes by following the relationships.

Graph database is extremely fast because of the localized nature of the graph traversal. Irrespective of the number of nodes and relationships in the database, the traversal only visit those nodes that are connected to the starting nodes. In a relational database, the join operations compute the Cartesian product and then discard irrelevant results which affect the performance exponentially with the growth of data set.

As an example, let say in a small stadium, if I am asked to count all the people sitting within 15 meters of me, I will stand and count all people sitting within the distance. I would not care how many people are there in the stadium. I can do the same task in a large stadium at a very similar time as I am not considering the size of the people in the stadium but the people near me. This is exactly how graph traversal works.

On the other hand, if the stadium example is taken for a relational database, it will count all the people in the stadium and discard those who are not sitting within 15 meters which is definitely not an effective strategy.

In a graph database, as the name suggests, data are stored as graph. A graph is a set of nodes and relationships that connect them. A graph is very expressive for representing real-world data as if we are writing them on white boards. The entities are represented as nodes in a graph and how entities are related to each other is represented by edges. For example, a data of a social network can be represented using a graph as shown in Figure 2.1



**Figure 2.1:** Representation social graph

Each node has a label that differentiates and group the nodes from other types of nodes. The nodes are connected with a directed edge that gives a semantic context like Ruth follows Harry.

Graph databases use graph data structure to store and query. We studied a very popular graph database called **Neo4j** that uses Labeled Property Graph (LPG) for representing graph data. RDF graph is another type of graph used to represent RDF triples. We discussed RDF graph in Semantic Web section above.

### 2.2.1 LPG - Label Property Graph

As the name suggests, a label property graph is made up of nodes, relationships, properties, and labels. Nodes in LPG can have one or more labels. Labels group the nodes together and indicate the roles these nodes play. For example, nodes in a graph can represent User, Order or Employee with a proper label.

The nodes in LPG has an internal structure and they contain properties as key-value pairs. We can think the node in LPG as an object and the properties as its attributes. For example, a User node can contain the properties like name, age, date of birth etc.

The relationships connect the nodes. A relationship always has a name, a start node and end node and is always directed from start node to end node. Like the nodes, relationships can also have properties. For example, a *FOLLOWS* relationship shown in Figure 2.1 can have a date property which add more semantic to the relationships. In LPG, there cannot have any broken link. Since a relationship always has a start and end node, we cannot delete a node without deleting its relationships.

### 2.2.2 Index Free Adjacency

In a graph database, every node stored in the database has a pointer to its adjacent nodes. Hence, to find the neighbors of a node, we do not need any additional helper structure such as indexing. By storing the data in this manner, graph databases are able to follow the pointer to connected nodes and relationships very quickly when performing traversals.

For example, to see how many users are followed by Ruth in our social graph example, we just need to find the Ruth node, then from there we can traverse to all nodes that have a follows relationship in a constant time. Nodes in a graph database can be indexed using properties and label which facilitates the task of finding the starting node for traversal.

### 2.2.3 Graph Traversal

Graph traversal is a key ingredient of graph databases. All the graph queries use graph traversal for retrieving information from a database. Traversing a graph means visiting its nodes following relationships according to some rules.

If we want to find all Ruth's friends, the traversal with start from the Ruth node and follow all *outgoing* FRIENDS relationship and get the end nodes.

If we want to find the friend-of-friend of Ruth, we start from Ruth node and navigate to all outgoing FRIENDS relationship and from each end node, we follow again outgoing FRIENDS relationship and get the end nodes. Similarly, we can do the traversal for the entire graph or for dept n. Friend-of-friend is a depth-2 traversal.

Graph databases use the index for locating the starting node. Like a relational database, an index can be created using properties such as name, email etc. After locating the starting node, the graph database does not need any indexing for traversal because of the index-free adjacency structure.

The index-free adjacency makes the graph traversal extremely fast compare to non-index-free adjacency store such as a relational database.

Depth-first-search and Breadth-first-search are two basic types of graph search algorithms.

#### 2.2.4 Graph Query

**SPARQL** is a standardized query language for a triple store. For other graph databases like Neo4j, vendor-specific query language exists. The query language used by Neo4j is called **Cypher**.

The graph queries can be represented by a graph where nodes or relationships can be variables. The graph database performs a graph matching with the stored graph and the query graph then returns the results.



## Chapter 3

# Extracting information from Excel file for building graph

In this chapter, we discuss extracting information from Excel files for building the graph that we will ultimately use to recognize entities and inferring new knowledge for the recognized entities. We describe the structure of the files that are provided by **Loops** and developed a system called **object extractor** that extracts the relevant information from the files.

### 3.1 Structure of the files

**Loops** gives us the dataset required for this research. The data they provide us are in MS Excel format. Each Excel file has tens of Excel Sheets having thousands of rows in each sheet. Though the dataset is semi-structured, it is not ideal to use directly in the graph as there are so many unnecessary, redundant and conflicting data. For a graph, we need to extract the information for nodes and relationship between nodes. In *Loops* dataset, we find that there can be two different type for Excel sheet which we refer as **Type I** and **Type II** accordingly. These are shown in Figure 3.1 and 3.2 respectively. As we said earlier, an Excel file can have tens of sheets but not all the sheets are relevant for the graph. In the Excel file for this thesis, all the sheets that have sheet name start with **Cluser** are considered to be relevant sheet and we only read those sheets for extracting information for the graph.

H	I	J	K	L	M	S	AD	AE
obj_asset	obj_field	obj_well			alt_field_1	obj_license		
SAP Asset code	Field	Wells in field	Well status	Legal entity	BoR Naming	Production Lic	Entry Point	Node
	Grand Total	GRAND TOTAL	Grand Total		GRAND TOTAL	GRAND TOTAL	Grand Total	
D12A	D12-A (Field)	D12-A1	Producing	GDFPN	D12-A	D12A	D15a-A (Facility)	D15a-A (Facility)
D12A	D12-A (Field)	D12-A2	Producing	GDFPN	D12-A	D12A	D15a-A (Facility)	D15a-A (Facility)
D12A	D12-A (Field)	D12-A3	Producing	GDFPN	D12-A	D12A	D15a-A (Facility)	D15a-A (Facility)
D12A	D12-A (Field)	D12-A1	Producing	PARNE	D12-A	D15 MAIN	D15a-A (Facility)	D15a-A (Facility)
D12A	D12-A (Field)	D12-A2	Producing	PARNE	D12-A	D15 MAIN	D15a-A (Facility)	D15a-A (Facility)
D12A	D12-A (Field)	D12-A3	Producing	PARNE	D12-A	D15 MAIN	D15a-A (Facility)	D15a-A (Facility)
D15A	D15a-A (Field)	D15-A101	Producing	GDFPN	D15-A	D12A	D15a-A (Facility)	D15a-A (Facility)
D15A	D15a-A (Field)	D15-A102	Shut-in	GDFPN	D15-A	D12A	D15a-A (Facility)	D15a-A (Facility)
D15A	D15a-A (Field)	D15-A101	Producing	PARNE	D15-A	D15 MAIN	D15a-A (Facility)	D15a-A (Facility)
D15A	D15a-A (Field)	D15-A102	Shut-in	PARNE	D15-A	D15 MAIN	D15a-A (Facility)	D15a-A (Facility)
D151	D15a-A104	D15-A104	Shut-in	GDFPN	D15-A104	D15 MAIN	D15a-A (Facility)	D15a-A (Facility)
D151	D15a-A105	D15-A105	P&A	GDFPN	D15-A104	D15 MAIN	D15a-A (Facility)	D15a-A (Facility)
E17A	E17a-A (Field)	E17A-A1	Shut-in	GDFPN	E17A-A	E17AB	E17a-A (Facility)	E17a-A (Facility)
E17A	E17a-A (Field)	E17A-A2	Producing	GDFPN	E17A-A	E17AB	E17a-A (Facility)	E17a-A (Facility)
E17A	E17a-A (Field)	E17A-A3	Producing	GDFPN	E17A-A	E17AB	E17a-A (Facility)	E17a-A (Facility)
E17A	E17a-A (Field)	E17A-A4	Producing	GDFPN	E17A-A	E17AB	E17a-A (Facility)	E17a-A (Facility)
E17A	E17a-A (Field)	E17A-A5ST1	Producing	GDFPN	E17A-A	E17AB	E17a-A (Facility)	E17a-A (Facility)
F3LG	F3-LGMG (Field)	F3-B101	Shut-in	GDFPN	F3-LGMG	F3B	F3-B (Facility)	F3-B (Facility)
F3LG	F3-LGMG (Field)	F3-B102	Producing	GDFPN	F3-LGMG	F3B	F3-B (Facility)	F3-B (Facility)
F3LG	F3-LGMG (Field)	F3-B103	Shut-in	GDFPN	F3-LGMG	F3B	F3-B (Facility)	F3-B (Facility)
F3LG	F3-LGMG (Field)	F3-B104	Producing	GDFPN	F3-LGMG	F3B	F3-B (Facility)	F3-B (Facility)
F3LG	F3-LGMG (Field)	F3-B105	Producing	GDFPN	F3-LGMG	F3B	F3-B (Facility)	F3-B (Facility)
F3LG	F3-LGMG (Field)	F3-B106	Shut-in	GDFPN	F3-LGMG	F3B	F3-B (Facility)	F3-B (Facility)
F3LG	F3-LGMG (Field)	F3-B109	Producing	GDFPN	F3-LGMG	F3B	F3-B (Facility)	F3-B (Facility)
F3UG	F3-UG (Field)	F3-B107	Shut-in	GDFPN	F3-UG	F3B	F3-B (Facility)	F3-B (Facility)
F3UG	F3-UG (Field)	F3-B108	Producing	GDFPN	F3-UG	F3B	F3-B (Facility)	F3-B (Facility)
F16E	F16-E (Field)	E16-A1	Producing	GDFPN	E18-F16	F16/E18A	F16-A (Facility)	F16-A (Facility)

Figure 3.1: A sample Excel sheet with data to be extracted (Type I)

Worksheet Type	Country	Object Type	Object Name	Alternate_1	Alternate
		Country	United States of America	US	USA
		Country	United States Minor Outlying Islands	UM	UMI
		Country	Uruguay	UY	URY
		Country	Uzbekistan	UZ	UZB
		Country	Vanuatu	VU	VUT
		Country	Venezuela	VE	VEN
		Country	Viet Nam	VN	VNM
		Country	Virgin Islands (British)	VG	VGB
		Country	Virgin Islands (U.S.)	VI	VIR
		Country	Wallis and Futuna	WF	WLF
		Country	Western Sahara	EH	ESH
		Country	Yemen	YE	YEM
		Country	Zambia	ZM	ZMB
		Country	Zimbabwe	ZW	ZWE
	Algeria	License	SE ILLIZI		South East Illizi
	Algeria	License	TOUAT		
	Australia	License	NT/RL-1		NT/RL1
	Australia	License	WA-27-R		WA27R
	Australia	License	WA-40-R		WA40R
	Australia	License	WA-6-R		WA6R
	Azerbaijan	License	ABSHERON		
	Brazil	License	BT-PN-3		BTPN3
	Brazil	License	PN-T-101		PNT101
	Brazil	License	PN-T-103		PNT103
	Brazil	License	REC-T-145		RECT145
	Brazil	License	REC-T-225		RECT225
	Brazil	License	REC-T-239		RECT239
	Brazil	License	REC-T-240		RECT240
	Brazil	License	REC-T-253		RECT253

Figure 3.2: A sample Excel sheet with data to be extracted (Type II)

### 3.1.1 Structure of Type I sheet

In Type I, there are two header rows. The first header row is what we call as *notation header*. The second header is a general header. If we have notation header in a sheet, the second header is not relevant. It is important to notice that the *notation header* is not applied to all columns in the sheet. Hence, we can discard the columns that do not have *notation header* as these columns are not going to be part of the relevant information.

In Type I sheet, we also have to pay attention to the names of the notation header. The name of the notation can be of two types:  $obj\_X$  and  $alt\_X\_Y$  where  $X$  is a type of a possible node and  $Y$  is a number. If the notation begins with  $obj\_X$ , we treat each cell value in that column as the name of an object with type  $X$ . For example, for the



column in the Figure 3.1 that has the notation, *obj\_well*, the cell value *D12-A1* under that column is a name of the object and the type of that object is *well*.

Similarly, if the name of the notation header has the form of *alt\_X\_Y*, we treat the column as an alternative name for the object type X. Y denotes the number of the alternative name as an object can have multiple alternative names. For instance, *alt\_field\_1* notation means that the cell values for the column with this notation header are alternative names for the object type *field*. *alt\_X\_Y* will always be accompanied by *obj\_X* header as an alternative name of an object without the object itself is meaningless.

### 3.1.2 Structure of TYPE II Sheet

For Type II, unlike Type I, there is no *notation header* but a single *general header* as shown in Figure 3.2. Depending on the name of the header, we can guess if a column in this file is relevant or not. If the header contains **Object Type** followed by **Object Name**, we can assume that this sheet has relevant information. As the name suggest, *Object Type* column has the type of the objects such as Country, License etc whereas *Object Name* column has the name of the objects such as *Norway*, *D12-A1* etc.

In addition to the said headers, there could be other relevant headers that have the format of **Alternate\_Y**. The columns with *Alternate\_Y* are the alternative names of the objects. Similar to Type I, *Alternate\_Y* will always accompanied by *Object Type* and *Object Name* headers as alternate names cannot exist without the object.

## 3.2 Extracting the Objects

Now we know the format of the Excel file and we can extract the information related to the object. We can take the object in the same context as the object in Object Oriented Programming (OOP). In our object, we have two *mandatory* attributes or properties. These are *name* and *type*. There is another optional property which is *alternative names*. The combined attributes of *name* and *type* of the object makes the object unique among other objects. Hence, we can assume that there will be no two objects with the same *name* and *type*.

A Java program is developed to extract the information from the Excel file. The program uses *Apache POI* library to manipulate Excel files. The program scans all the relevant sheets that have names start with **Cluster**. For each such relevant sheet, the program reads the first row to extract the header. If the header is *notation header*, i.e at least one of the cell values for the header starts with *obj\_*, then the program discards the second

row which is a general header. As we said earlier, if a sheet contains *notation header*, the general header is not relevant. An abstract of the header extraction is shown in Listing 3.1

```

1  public static List<String> getHeader(Iterator<Row> rowIterator) {
2      List<String> resultHeader = new ArrayList<>(50);
3      Row heading = rowIterator.next();
4      boolean removeNextHeading = false;
5
6      for (int i = 0; i < heading.getLastCellNum(); i++) {
7          Cell cell = heading.getCell(i);
8          if (cell == null) {
9              resultHeader.add(" ");
10             continue;
11         }
12         String val = cell.getStringCellValue();
13         if (val.startsWith("obj_")) {
14             removeNextHeading = true;
15         }
16
17         resultHeader.add(val);
18     }
19
20     if (removeNextHeading) {
21         rowIterator.next(); // removes general header
22     }
23     return resultHeader;
24 }

```

**Listing 3.1:** Header extraction

After extracting the header, the program read each row and scan the relevant cell values. Each valid row has several cells and the combination of the relevant cells in a row forms an object. A cell `cell[i]` in *row* is relevant if the corresponding header satisfies one of the following rule:

- header[i] starts with *obj\_*
- header[i] starts with *alt\_*
- header[i] starts with *Object Name* and header[i-1] starts with *Object Type*
- header[i] starts with *Alternate*

The value of the cell can be *name*, *type* or *alternative name* of an object based on the corresponding header. For example, if the corresponding *header[i]* has format of *obj\_X*,

then the cell value is name of an object. The type of the object is X. If the header is of the format *alt\_X\_Y*, then the cell value is an alternative name of the object whose type is X. The corresponding object name can be read from the cell as *cell[header.indexOf(X)]*. The objects extraction is illustrated in Figure 3.3. The redundancy of the object names such as multiple *D12-A (Field)*, *D12A* are handled with a clever selection of data structure to hold the objects.

	I	J	K	L	M	S
	obj field	obj well			alt field 1	obj license
	Field	Wells in field	Well status	Legal entity	BoR Naming	Production Lice
	D12-A (Field)	D12-A3	Producing		D12-A	D12A
	D12-A (Field)	D12-A3	Producing		D12-A	D15 MAIN
	D15a-A (Field)	D15-A101	Producing		D15-A	D12A

```
header = ['obj_field', 'obj_well', "", "", 'alt_field_1', 'obj_license']
```

```
objects = {
    'field': { 'D12-A (Field)': ['D12-A'],
              'D15a-A (Field)': ['D15-A']
            },
    'well': { 'D12-A3': [],
              'D15-A101': []
            },
    'license': { 'D12A': [],
                 'D15 MAIN', []
               }
}
```

Figure 3.3: Illustration of objects extraction

### 3.2.1 Data Normalization

For each cell value we extracted, we need to perform data normalization by replacing *accent characters* and *Norwegian characters* with appropriate major character. For example, the word **Gjøa** needs to be transformed into **Gjoa**. We need to convert all the character into lower letters so that the objects **SE Illizi** and **SE ILLIZI** are treated as a single object. The following Listing 3.2 shows the data normalization.

```

1  private String normalizeText(String in) {
2
3  String acc = "ÀàÁáÂâÏóÔôÂâ"; // all possible accent characters
4  String maj = "AaAaAaOoAa"; // equivalent major characters
5
6  String out = "";
7  for (int i = 0 ; i < in.length() ; i++) {
8      String car = in.substring(i, i+1);
9      int idx = acc.indexOf(car);
10
11     if (idx != -1){
12         out += maj.substring(idx, idx+1);
13     } else if (car.equals("Æ")) { // combination of multiple characters
14         out += "AE";
15     } else if (car.equals("æ")) {
16         out += "ae";
17     } else {
18         out += car;
19     }
20 }
21
22 return out.toLowerCase(); // convert to lower case and return
23 }

```

Listing 3.2: Data normalization

### 3.3 Extracting the Relationships

After extracting the objects and their associated attributes, we are halfway down to build our graph. In a graph, we can represent the objects we extracted as nodes. However, another major ingredient for a graph is missing. A graph consists of a set of nodes and a set of edges or relationships that connects two nodes. We at the moment do not have the relationships. A relationship creates a link between two objects. Each relationship has a label. The label usually has the format of **startNodeType\_has\_endNodeType**. How two objects linked with each other is encoded in *notation header*. If we have **n** objects in **header**, number of relationship types we can have is given by:

$$C(n, k) = \frac{n!}{k!(n-k)!}$$

where  $k = 2$ .

For example, if we have the following header:

$$header = [obj\_field, obj\_license, obj\_asset]$$

The type of relationship we can have are:

$$field\_has\_license, field\_has\_asset, license\_has\_asset$$

The extraction of relationship is illustrated graphically in Figure 3.4. In the figure, we have three headers that start with *obj\_*. Hence, according to the formula, we will have  $\frac{3!}{2!*1!} = 3$  relationship types. The datatype we selected to store the relationship effectively removes redundant relationships. In the figure, there are multiple similar relationships such as *D12-A (Field) to D12-A3* appears two times but the program only stores one such relationship.

I	J	K	L	M	S
obj_field	obj_well			alt_field_1	obj_license
Field	Wells in field	Well status	Legal entity	BoR Naming	Production Lice
D12-A (Field)	D12-A3	Producing		D12-A	D12A
D12-A (Field)	D12-A3	Producing		D12-A	D15 MAIN
D15a-A (Field)	D15-A101	Producing		D15-A	D12A

```
header = ['obj_field', 'obj_well', "", "", 'alt_field_1', 'obj_license']
```

```
relations = {
    'field_has_well': [
        ['D12-A (Field)', 'D12-A3'],
        ['D15a-A (Field)', 'D15-A101']
    ],
    'field_has_license': [
        ['D12-A (Field)', 'D12A'],
        ['D12-A (Field)', 'D15 MAIN'],
        ['D15a-A (Field)', 'D12A'] ],
    'well_has_license': [
        ['D12-A3', 'D12A'],
        ['D12-A3', 'D15 MAIN'],
        ['D15-A101', 'D12A']
    ]
}
```

**Figure 3.4:** Illustration of relationships extraction

### 3.3.1 Data Clean up

As we mentioned earlier, the Excel file has redundant and ambiguous data that we need to clean up after extraction. The ambiguity will appear when a value appears both as object and alternative names. For example, in one of the Excel sheet, **UK** is an alternative name of a *country* type object whose name is **United Kingdom** but in another sheet, **UK** appears as the name of the country object. If we do not clean up this type of ambiguity, we will have multiple nodes in a graph that represent the same object in real world. Some of the data ambiguity and their cleanup operation is shown below.

#### Case 1:

Let say we have the following two country type objects with their alternative names.

$$UnitedKingdom = [UK]$$

$$UK = []$$

Since **UK** is alternative name of **United Kingdom**, we can infer that the *UK* and *United Kingdom* are same country, hence, we cannot have *UK* as separate country. So we have to merge the two into  $UnitedKingdom = [UK]$

#### Case 2:

Let say we have the following two country type objects and their alternative names.

$$Netherlands = [NL, NLD]$$

$$TheNetherlands = [NLD]$$

Since the alternative names of the two objects have a similar item (NLD), we can infer that the two objects are same. So we have to merge them into one. The final object will be:

$$Netherlands = [NL, NLD, TheNetherlands]$$

#### Case 3:

Let say we have the following objects:

$$F3 = [F03, F03B, F03b]$$

$$F3\_1 = []$$

And we have a relationship as:

$$F03b \rightarrow F3\_1$$

Since, **F03b** is an alternative name of object **F3**, we have to change the relationship into:

$$F3 \rightarrow F3\_1$$

Also, the name of the object in one sheet can appear as an alternative name of that very same object. Having the same value for an object's name and its alternative name is a clear redundancy. We need to clean up this type of redundancy as well. Furthermore, there is some inconsistency in data. For example, there are some alternative names whose object's names are *null* or *empty*. An alternate name without an associated object is a clear violation of the rule, hence we need to clear such data inconsistency as well. Finally, not all the cells have valid values. For example, some of the cells have **N/A**, **None**, **(none)**, - etc. We need to take care of the rows that have invalid cell values.

### 3.4 Saving Objects and Relationships for Future use

From the full dataset provided by LOOPS, we extracted **10** types of objects and **23** different type of relationships among them. We need to save the extracted objects and their relationships for building the graph in a later stage. The extracted information are saved as structured Excel file where we have several sheets. Each sheet represents a type of object or a type of relationships depending on the name of the sheets. The object sheet has name that starts with **obj\_X** where **X** is the type of the object. Similarly, the relationship sheet name start with **rel\_X\_has\_Y** where **X** and **Y** are two object types. For instance, if we have *country* and *license* type objects, we are effectively saving all the *country* type and *license* type objects into *obj\_country* and *obj\_license* sheets respectively and the relationship from *country* to *license* is saved in *rel\_country\_has\_license* sheet. Figure 3.5 and 3.6 show the structure of the saved objects and relationships respectively.

### 3.5 Experimental Setup and Result

For evaluation of the *object extractor* system, we need a dataset with the known number of objects and relationships. Unfortunately, for this experiment, LOOPS doest not provide us a dataset with the known number of objects and relationships. Therefore, we make a sample dataset by taking a *Type I* and a *Type II* sheets from the original dataset. Then we use *object extractor* to extract objects and relationships. We manually cross-check the

Object Name	Alternate Names
bentheim a	
hamm-sud	
bentheim c	
hummling	
bentheim b	
pl578	pl 578,578,tower (pl153),578.0
wettrup-holte a	
dns-b-20-008/52 (b/c)	dns b 20 008/52,dns b 20 008/52 (b/c)
e15c	
e15a	
southern north sea (sheringh	sheringham west prospect
d18a	d18-a orca
absheron	(az) absheron
struktur fuerstenwalde	
pl107	njord future project (nfp),132,107.0,pl 132,107,pl132,132.0,njord area,pl 107
obermehler	
pl348	pl 348,348,348.0
nl assets (rm)	
pl100	110b,pl110,pl099,pl077,pl078,snohvit unit,pl097,097,448.0,110,099,077,078,pl 100,pl 064,64.0,110.0,1
southern north sea (galileo p	galileo prospect, cavendish discovery
wellmitzer laqune	
wenze	
pn-t-101	pn t 101,(br) pn-t-101,pnt101
ruedersdorf	
pl630bs	pl 630 bs
pn-t-103	pn t 103,pnt103,(br) pn-t-103
n07b	n7b
schiermonnikoog-noord	

Figure 3.5: Structure of object sheet

From	To
wa-27-r	tern
wa-6-r	petrel
nt/r1-1	petrel
wa-40-r	frigate
muara bakau	muara bakau
arguni i	arguni i
north ganal	north ganal
block 2f	deepwater block 2f sarawak
block 3f	deepwater block 3f offshore sarawak
p2135	d12-a
d15	d15-a, d15-a-104, d15-tourmaline
d18a	d18-fa
e17a & e17b	e17-a
e18a	e18-a
f03b	f3-fb
f16	f16-p
q14 & q17b	q14-a/b, q14-c
q16a	q16a-a, q16a-b
q17a	q17a-s1
q17c & q17d	q17cd-a
k03a	k2b-a
k09a & k09b	k9ab-a, k9ab-b
k09c	k9c-a, k9c-b, k9c-c
k12	k12-a, k12-b, k12-b-09, k12-c, k12-d, k12-e, k12-q, k12-s1, k12-s2, k12-s3
l05a	l5a-a, l5a-d
l10 & l11a	l10 central dev. area, l10-19, l10-6, l10-g, l10-k, l10-m, l10-s1, l10-s2, l10-s3, l10-s4, l11-1, l11-7, l11a-a, l11-lark
l12a	l12a-b, l12-fa
l12b & l15b	l12-fc, l15-fa

Figure 3.6: Structure of relationship sheet

number of objects and relationships in the sample dataset with the number of objects and relationships extracted by the system. The information about the sample dataset and the number of objects extracted by *object extractor* is given in Table 3.1.

Objects	Number of Objects	Unique Objects	Extracted Objects	Remark
Country	1690	250	250	100%
License	1562	400	393	98%
Unit	115	14	14	100%
Block	92	92	92	100%
Pipeline	2	2	2	100%

Table 3.1: object extraction by object extractor



In the sample dataset, there are **five** different types of objects: *Country*, *License*, *Unit*, *Block* and *Pipeline*. The *object extractor* system extracts all unique Country, Unit, Block and Pipeline objects without any miss (100%). In case of the license, there are 400 unique license objects, however, the system extracts 393 of them. So we take a deep analysis for License objects along with their alternative names. It is found that, for the license objects, there are some objects that share common alternative names. When two objects having similar type have one or more common alternative names, those objects are considered to represent the same object and hence are merged into a single object. Therefore, the number of extracted license objects is less than the number of unique license objects in the sample dataset.

In the sample dataset, there are three type of relationship: *country\_has\_license*, *country\_has\_unit* and *license\_has\_unit*. The *object extractor* system extracts all of the relationships accurately.

## 3.6 Conclusion

The *object extractor* extracts objects, their alternative names and the relationship among the objects from Excel file. The system also normalizes the extracted information. It handles data redundancy and data ambiguity as well. The system saves the extracted information into the structured file that can be used directly to build the graph.

In the next chapter, we use the information we have extracted from **LOOPS** dataset and represent them as a graph. Furthermore, we use the graph for recognizing entities and inferring new information that is hidden in the dataset.



## Chapter 4

# Implementation of Graph for Entity Recognition and Inference

### 4.1 Introduction

In the previous chapter, we build a system that extracts information from Excel files provided by **LOOPS**. The system basically extracts two types of information: *objects* and *relationships*. In this chapter, we discuss different types of graphs and their representation and we choose a suitable graph to represent the information we extracted. Finally, we build a system called *GER* that recognizes the entities in an unseen dataset and infer more information about the recognized entities using graph theories.

### 4.2 Type of Graphs

Formally a graph is represented as  $G = (V, E)$  where  $V = v_1, v_2, \dots, v_n$  is a finite set of nodes and  $E \subseteq V \times V, E = (v_i, v_j), i \neq j$  represents the set of edges in the graph.

Graphs can be categorized in different ways based on their edge types, orientation, the presence of weights etc.

**Directed and Undirected Graphs:** A graph  $G$  is directed if there is a direction between the nodes in edges. Each edge in a directed graph has a *start* node and *end* node and edge  $e(v_i, v_j)$  is not similar to edge  $e(v_j, v_i)$ . On the other hand, the nodes in an edge of an *undirected* graph do not have a direction, hence edge  $e(v_i, v_j)$  is symmetric to edge  $e(v_j, v_i)$ .

**Labeled and Unlabeled Graph:** A graph  $G$  is a *labeled* graph if the nodes and/or the edges are labeled with some data in addition to the data that identifies the node and the edge. Otherwise, the graph is an unlabeled graph. If only the nodes have the labels, it is called *node labeled graph* and if labels are only for edges, it is called *edge labeled graph*.

**Wighted and Unweighted Graph:** A graph  $G$  is a weighted graph if the graph is an *edge labeled graph* and the labels can be operated on by the usual arithmetic operators, including comparisons like using less than and greater than.

**Simple Graph:** A simple graph is an undirected, unweighted graph where no self-loops or multiple edges between the same pair of nodes is allowed. If the graph is directed, it is called simple directed graph.

**Multigraph:** A *multigraph* is a graph where multiple edges between two nodes are allowed. If the graph is a directed graph, is called directed multigraph.

**Pseudograph:** A *pseudograph* is a graph that allows multiple edges and self loop. A pseudograph can be directed or undirected.

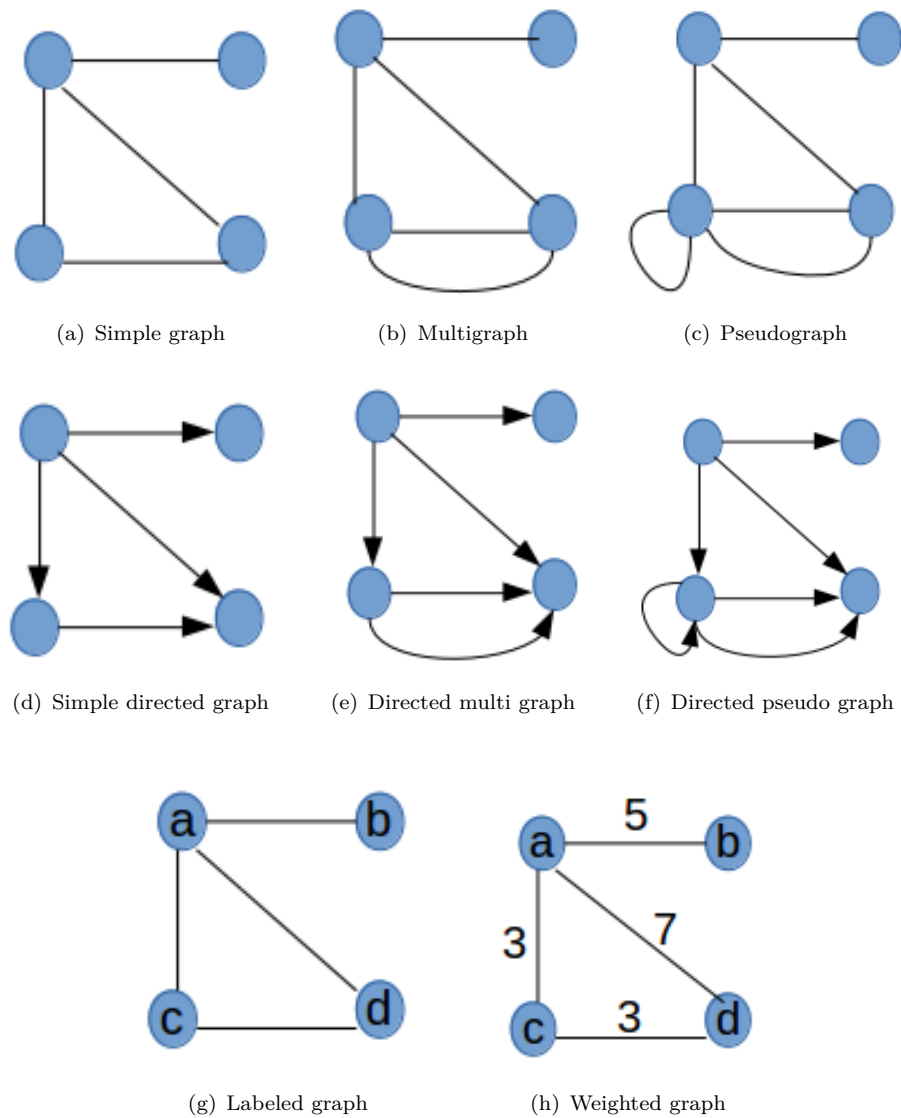
Figure 4.2 shows such graphs.

### 4.3 Representation of graphs

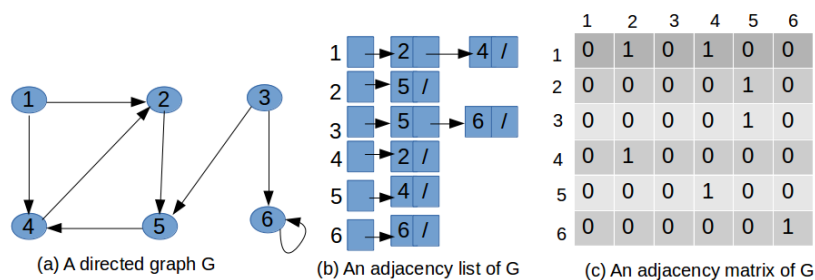
There are two standard ways to represent a graph  $G = (V, E)$ : as a collection of adjacency list or as an adjacency matrix. If the graph is *sparse* graph -those for which  $|E|$  is much less than  $|V|^2$  - adjacency list is the method of choice as it takes less space. An adjacency matrix is preferable if the graph is *dense* -  $|E|$  is close to  $|V|^2$ - or when we need to be able to tell quickly if there is an edge between two nodes [6]. Figure 4.2 shows two representation of a directed graph  $G$ .

### 4.4 Representing LOOPS data in graph

In the previous chapter, we extracted objects and relationships from the Excel file and save the information into a structured Excel file so that we can represent the extracted information in a graph. In above sections, we discuss different type of graphs and their representation using adjacency list and adjacency matrix. Having the data on hand, now we can choose a suitable graph that can represent the LOOPS data.



**Figure 4.1:** Different type of graphs



**Figure 4.2:** Representation of graph

After analyzing the relationships in LOOPS data, we find that there are no self-loops or parallel edges between two objects. Furthermore, the relationships have defined start and end such as *country\_has\_license*, *license\_has\_unit* etc. Hence, the most suitable graph for LOOPS data is *simple directed graph*.

Here we explain some important vocabularies that we often use for a directed graph. For example, in a directed graph  $G$ , if we have a relationships from  $nodeU$  to  $nodeV$ , then:

- $nodeU$  is a **predecessor** of  $nodeV$
- $nodeV$  is a **successor** of  $nodeU$
- $edgeUV$  is an **outgoing** edge for  $nodeU$
- $edgeUV$  is an **incoming** edge for  $nodeV$
- $nodeU$  is a **source** of  $edgeUV$
- $nodeV$  is a **target** of  $edgeUV$

For building the graph, we use *Google Guava* library written in Java that comes with common utility methods for adding or removing nodes and edges, getting predecessor and successor etc.

The node is represented as a Java class. The representation of the Node along with it's attributes is shown in Listing 4.1.

```
1 Class Node {  
2     String name;  
3     String node_type;  
4     String Set<String> alternative_names;  
5     // add other properties here ...  
6 }
```

**Listing 4.1:** Representation of node in Java

In Listing 4.2, the process of building graph using Guava is shown as *pseudocode*.

```

1 // initialize Guave Value Graph
2 Graph G = new ValueGraph()
3 // add nodes to the graph
4 forEach object_sheet in ExcelFile {
5     // object_sheet name has format: obj_X , where X is object type
6     // eg: obj_country, obj_license etc
7     node_type = object_sheet.name().replace("obj_", "");
8     forEach row in object_sheet {
9         //row[0] = name, row[1] = alternative names
10        Node node = new Node(row[0], node_type, row[1])
11        G.addNode(node); // add node to graph
12    }
13 }
14
15 // add relationships between nodes
16 forEach rel_sheet in ExcelFile {
17     rel_name = rel_sheet.name().replace("rel_", "")
18
19     // relationship sheet name has format: rel_X_has_Y, wher X and Y are object types
20     // eg: rel_country_has_license, rel_license_has_unit etc
21     fromType, toType = rel_name.split("_has_")
22     forEach row in rel_sheet {
23         // find the node in G for given name and type
24         // row[0] = fromNode name, row[1] = toNode name
25         fromNode = findNodebyNameAndType(row[0], fromType);
26         toNode = findNodebyNameAndType(row[1], toType);
27
28         if fromNode AND toNode NOT NULL {
29             G.putEdgeValue(fromNode, toNode, rel_name)
30         }
31     }
32 }

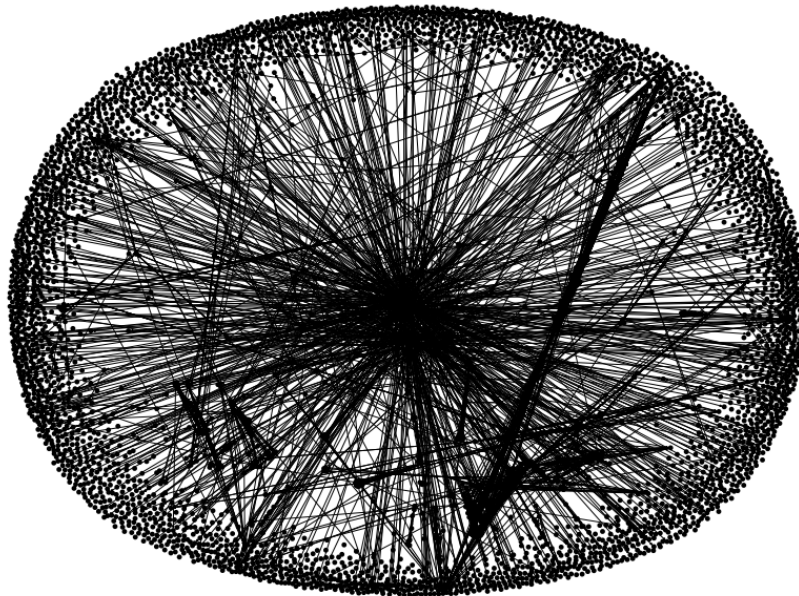
```

**Listing 4.2:** Pseudocode for building graph

Figure 4.3 visualize the LOOPS data in a graph.

## 4.5 Graph-based Entity Recognition

Entity Recognition (ER) is a hot topic among natural language processing group. A substantial amount of work has been done for recognizing Named Entities from the text. Named Entity Recognition (NER) technology recognizes proper nouns (entities) such as *location, person name, date, address etc* and associate them with appropriate types. The entities in Natural languages are sometimes pretty ambiguous. For example, a text



**Figure 4.3:** Visualization of LOOPS data in a graph

containing *Washington* may refer to US president George Washington or Washington DC depending on the surrounding context.

However, it is important to note that the entity recognition we are interested in this thesis is not to be mistaken with the Named Entity Recognition. In this thesis, the set of vocabularies for the entities are subjected to the entities of Oil & Gas industries. All the entities in our thesis are the objects that belong to this specific area.

Unlike NER, the type of entities we are interested in is not the pronouns in natural language. Our entities are the type of objects in Oil & Gas sectors such as *Country objects*, *License object*, *Well objects* etc. All the entities in our domain are represented as nodes in a graph. If we are given a text, our task for the entity recognition is to find out the nodes from the graph whose names or alternative names match with the word or words from the text.

Now, we will formally describe how entity recognition in our domain works. All the objects and their relationships we extracted in the previous chapter are our entities and we treat them as ground truth. We represent our ground truth in a graph. Now, suppose we are given a text containing **Drilling stop at F3-1**. Our task is to check and recognize if the text contains any entities that are in our ground truth. If we can recognize any entities, we need to report back with what are the recognized entities and what are their types. Although disambiguation is a crucial and challenging task in NER, we do not deal with the disambiguation in this research. If a text matches for two different entities with different types, we need to report both the entities and their types.



For instance, if in the given text **F3-1** matches with **F3 (License)** and **F3-1 (Well)**, the system need to give both as output.

Similar to the normalization we perform in object extraction, we need to normalize the text for the input file as well. We use the same normalization technique for normalizing input data.

From the implementation point of view, we already extracted our entities and build a graph with all the entities as nodes. Therefore, recognition of entities from a text should be straightforward. We have to iterate the nodes from the graph and match the text with the name and alternative names of the node to see if there is a match. If a match is found for a node, we can return the name and type of the node as a recognized entity. Listing 4.3 shows the pseudocode for entity recognition.

```

1  Function entityRecognition(G, filename) {
2      File inputFile = readExcelFile(filename) // read the input file
3      forEach row in inputFile {
4          forEach cell in row {
5              text = cell.value()
6              text = normalizeText(text) // normalize the text
7              // iterate each node in graph
8              forEach node in G.nodes() {
9                  if isContain(text, node.name) OR isContain(text, node.alternative_names()) {
10                     // save the recognized node to file
11                     addToResult(node)
12                 }
13             }
14         }
15     }
16 }

```

**Listing 4.3:** Pseudocode for Entity Recognition in a graph

## 4.6 Graph-based Inference

An inference is an idea or conclusion that's drawn from evidence and reasoning. An inference is an educated guess [7]. If we have an evidence, using that evidence, we can infer other knowledge. Ancient Greek philosophers defined a number of *sylogisms* that can be used as building block for reasoning. A famous example of inference is:

1. All humans are mortal.
2. All Greeks are humans.

3. Therefore, All Greeks are mortal.

In the above section, we recognize entities in a file using the graph. By taking the motivation from above *sylogisms*, we can infer more knowledge about the recognized nodes. For example, if we get a string **Gunnild went to Oslo last week**, and we have a graph like:

$$\text{Norway} \xrightarrow{\text{hasCapital}} \text{Oslo}$$

We can infer that Gunnild was in Norway last week as she was in Oslo which is the capital of Norway in our graph. However, if the text is **Gunnild went to Norway last week**, we cannot infer that she went to Oslo last week.

It is easy to retrieve the knowledge if two nodes are directly connected. However, in the graph, the two nodes may not be connected directly. The two nodes may be connected by multiple hops in the graph. Retrieving appropriate nodes via variable length paths relevant for entity recognition is a challenge.

We can overcome the challenge by using the power of the graph, especially by using graph traversal techniques. There exist two popular graph traversal algorithms. These are *Bread First Search* and *Depth First Search*.

**Bread First Search (BFS):** BFS starts at a node in a graph and explores the neighbor nodes first before visiting the next level neighbors. It can be used for both directed and undirected graph. A graph may contain cycles, so the algorithm may come to the same node again and again. To avoid processing a node more than once, the algorithm marks a visited node with a boolean flag.

**Depth First Search (DFS):** DFS starts at a node and explores as far as possible along each branch before backtracking. For instance, if the search starts at  $v$ , it visits one of  $v$ 's unexplored neighbors say  $u$ . From  $u$ , it continue the same process. If the algorithm reaches a node say  $x$  from where there are no reachable nodes, it backtracks to the node from where it was discovered. From that node, it visits one of the unexplored nodes if there are any, otherwise, it backtracks again. Similar to BFS, DFS can be used for both directed and undirected graph and it also uses a boolean flag to remember the visited nodes.

Which algorithm to choose heavily depends on the structure of the graph and location of the searched-for items. For example, if the solution is not far from the start node, then BFS might be better but if the search-for items are frequent and located deep in the graph, then DFS might perform well.

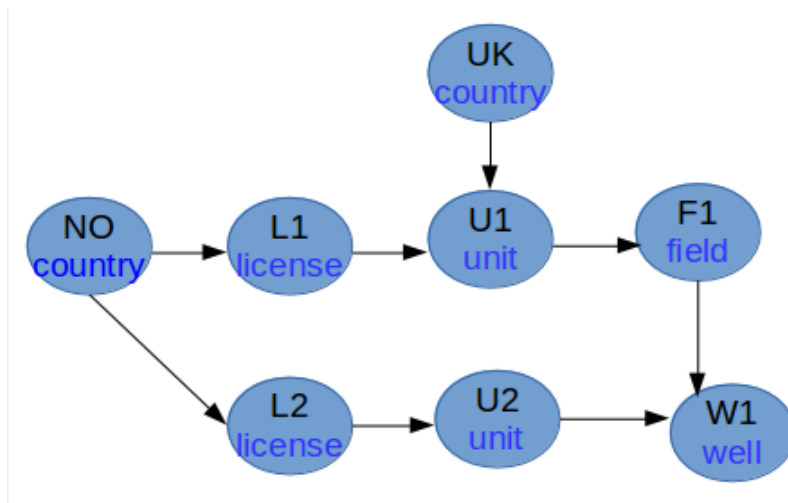
For the graph-based inference in this thesis, we use an algorithm that is similar to BFS. In BFS, the algorithm usually starts from a node and visits its successors. In other words, the algorithm visits from a parent node to its child nodes. However, for the inference, we need the opposite behavior. Therefore, we need to visit from a node to its predecessors.

We need to follow two rules while inferencing. The rules are defined as:

**Rule 1:** The inference algorithm should not infer more than one object with similar type.

**Rule 2:** If an object  $z$  infers two objects  $x$  and  $y$  and both  $x$  and  $y$  are of same type, then choose an object that is closer to  $z$  as result. If both  $x$  and  $y$  have same distance from  $z$ , then one object randomly.

Figure 4.4 shows a graph with 8 nodes. Each node has two fields: *the name* and *the type*. If we get a text : **W1, W2 and W3 exploration** and run the entity recognition algorithm discussed above with the text and the graph, the system will recognize **W1** (well).



**Figure 4.4:** Example of Inference

Now, we can use our inference algorithm to know more information about the recognized node **W1**. If we use the BFS in reverse order taking **W1** as starting node ( i.e traversing by following backward direction), the nodes we can explore from starting node are:

at 1 hops distance: **F1 and U2**

at 2 hops distance: **L2 and U1**

at 3 hops distance: **NO, UK and L1**

However, there are multiple nodes with the same type in the explored list of nodes such as [U1, U2], [L1, L2] and [NO, UK].

According to the rules, we cannot infer more than one object that are of the same type. Therefore, the inference algorithm will infer  $U2$  over  $U1$  as  $U2$  has a shorter distance to  $W1$ . Similarly,  $L2$  will be inferred instead of  $L1$ . For  $NO$  and  $UK$ , both nodes have same type and are at same distance from  $W1$ . Therefore, the first visited node will be selected as inferred node.

The set of inferred nodes will be :

$$F1(\text{field}), U2(\text{unit}), L2(\text{license}), NO(\text{country})$$

OR

$$F1(\text{field}), U2(\text{unit}), L2(\text{license}), UK(\text{country})$$

depending on the algorithm that visit  $NO$  or  $UK$  node first.

The steps for the inference algorithm are given in Listing 4.4.

```

1  function Inference(G, start) {
2      inferredNodes = {} // holds inferred nodes
3      Q = {}; // empty queue
4      ENQUEUE(Q, start) // add item to Q
5      start.visited = true; // set visited flag
6      while (Q is not empty) {
7          u = DEQUEUE();
8          // iterate each predecessor of node u
9          for each v in G.getPredecessors(u) {
10             if (v is not visited) {
11                 ENQUEUE(Q, v) // add v to Q
12                 v.visited = true; // set visited flag for v
13                 // add v to inferredNodes if type if v node is not present
14                 if not inferredNodes.haveNodeType(v.getType) {
15                     ENQUEUE(inferredNodes, v) // add node v to inferred node
16                 }
17             }
18         }
19     }
20 }
21 return inferredNodes;
22 }
```

**Listing 4.4:** Algorithm for Inference in a graph

## 4.7 Experimental Setup and Result

In this section, we evaluate **GER** system that we build for graph-based entity recognition and inference. For the experimentation of GER, LOOPS provides us with sample objects and relationships for the graph and a sample dataset for recognizing and inferencing entities in the dataset using the graph. There are 40 objects and 35 relationships. The sample dataset for entity recognition and inferencing is shown in Figure 4.5.

Processing Date	Task Name	Task Name Concatenated
20-Sep-2017	Drilling Planning Monthly February 2013	Drilling Planning Monthly February 2013
20-Sep-2017	F3-B106ST (D)	Drilling Planning Monthly February 2013 - F3-B106ST (D)
20-Sep-2017	L5a-D3 (D) Completion	Drilling Planning Monthly February 2013 - L5a-D3 (D) Completion
20-Sep-2017	K9-13 (D)	Drilling Planning Monthly February 2013 - K9-13 (D)
20-Sep-2017		Drilling Planning Monthly February 2013 -
20-Sep-2017	L10-C1 (P&A)	Drilling Planning Monthly February 2013 - L10-C1 (P&A)
20-Sep-2017	L10-C6 (P&A)	Drilling Planning Monthly February 2013 - L10-C6 (P&A)
20-Sep-2017	L10-C5 (P&A)	Drilling Planning Monthly February 2013 - L10-C5 (P&A)
20-Sep-2017	L10-C4 (P&A)	Drilling Planning Monthly February 2013 - L10-C4 (P&A)
20-Sep-2017	L10-C2 (P&A)	Drilling Planning Monthly February 2013 - L10-C2 (P&A)
20-Sep-2017	L10-C6 (P&A)	Drilling Planning Monthly February 2013 - L10-C6 (P&A)
20-Sep-2017	L10-C5 (P&A)	Drilling Planning Monthly February 2013 - L10-C5 (P&A)
20-Sep-2017	L10-C3 (P&A)	Drilling Planning Monthly February 2013 - L10-C3 (P&A)
20-Sep-2017	L10-C1 (P&A)	Drilling Planning Monthly February 2013 - L10-C1 (P&A)
20-Sep-2017	L10-C3 (P&A)	Drilling Planning Monthly February 2013 - L10-C3 (P&A)
20-Sep-2017	L10-C1 (P&A)	Drilling Planning Monthly February 2013 - L10-C1 (P&A)
20-Sep-2017	L10-C3 (P&A)	Drilling Planning Monthly February 2013 - L10-C3 (P&A)
20-Sep-2017	L10-C1 (P&A)	Drilling Planning Monthly February 2013 - L10-C1 (P&A)
20-Sep-2017	L10-F3ST1 (D)	Drilling Planning Monthly February 2013 - L10-F3ST1 (D)
20-Sep-2017		Drilling Planning Monthly February 2013 -
20-Sep-2017	Q13-A1 Snubbing	Drilling Planning Monthly February 2013 - Q13-A1 Snubbing
20-Sep-2017	Q13-A5ST1 Snubbing	Drilling Planning Monthly February 2013 - Q13-A5ST1 Snubbing
20-Sep-2017		Drilling Planning Monthly February 2013 -
20-Sep-2017	ROEB 5 (D/O)	Drilling Planning Monthly February 2013 - ROEB 5 (D/O)
20-Sep-2017		Drilling Planning Monthly February 2013 -
20-Sep-2017	ROEB 5 Stimulation & ESP	Drilling Planning Monthly February 2013 - ROEB 5 Stimulation & ESP
20-Sep-2017		Drilling Planning Monthly February 2013 -
20-Sep-2017	Norddeutschland 5 P&A	Drilling Planning Monthly February 2013 - Norddeutschland 5 P&A
20-Sep-2017		Drilling Planning Monthly February 2013 -

**Figure 4.5:** Sample dataset for entity recognition and inferencing

In the sample dataset, there are 425 rows and 12 columns. GER system reads each cell value of the dataset and recognizes the entities if there are any. If the system recognizes an entity, it tries to infer more entities for the recognized entity. The recognized and inferred entities are added to the dataset as new columns. Recognized entities have a header with format **erType** such as **erCountry** and the inferred entities have header with format **eiType** such as **eiCountry**. Figure 4.6 shows a sample of the entities recognized and inferred in the sample dataset.

The number of recognized and inferred entities in sample dataset is given in Table 4.1. We cross check the result with the provided solution set and found that GER system recognizes all the entities and inferred all the possible entities accurately.

## 4.8 Conclusion

GER system recognizes entities and inferred more entities for each recognized entities. We tested the system against a sample dataset and verify that the system is able to

Task Name	N	O	P	Q	R	S
	eiCountry	eiLicense	eiUnit	eiProject	eiBusinessare	erlicense
Drilling Planning Monthly February 2013						
F3-B106ST (D)	Netherlands	F3-				F3-
L5a-D3 (D) Completion	Netherlands	L5a				L5a
K9-13 (D)	Netherlands	K9-				K9-
L10-C1 (P&A)	Netherlands	L10				L10
L10-C6 (P&A)	Netherlands	L10				L10
L10-C5 (P&A)	Netherlands	L10				L10
L10-C4 (P&A)	Netherlands	L10				L10
L10-C2 (P&A)	Netherlands	L10				L10
L10-C6 (P&A)	Netherlands	L10				L10
L10-C5 (P&A)	Netherlands	L10				L10
L10-C3 (P&A)	Netherlands	L10				L10
L10-C1 (P&A)	Netherlands	L10				L10
L10-C3 (P&A)	Netherlands	L10				L10
L10-C1 (P&A)	Netherlands	L10				L10
L10-C3 (P&A)	Netherlands	L10				L10
L10-C1 (P&A)	Netherlands	L10				L10
L10-F35T1 (D)	Netherlands	L10				L10
Q13-A1 Snubbing	Netherlands	Q13				Q13
Q13-A55T1 Snubbing	Netherlands	Q13				Q13
ROEB 5 (D/O)						
ROEB 5 Stimulation & ESP						
Norddeutschland 5 P&A						
Ordovician Formation						
F3-B106ST (D)	Netherlands	F3-				F3-
L5a-D3 (D) Completion	Netherlands	L5a				L5a
K9-13 (D)	Netherlands	K9-				K9-

**Figure 4.6:** Sample of recognized and inferred entities

Objects	Recognized	Inferred
Country	157	260
License	130	7
Unit	40	4
Business Area	113	0
Project	8	0

**Table 4.1:** Result of recognized and inferred entities for sample dataset

recognize and infer all the entities that are in the graph. The system is able to recognize entities in a text irrespective of the formatting of the text. This is possible because of the normalization of the text before recognizing entities.

## Chapter 5

# Link Prediction - Unsupervised Approach

### 5.1 Introduction

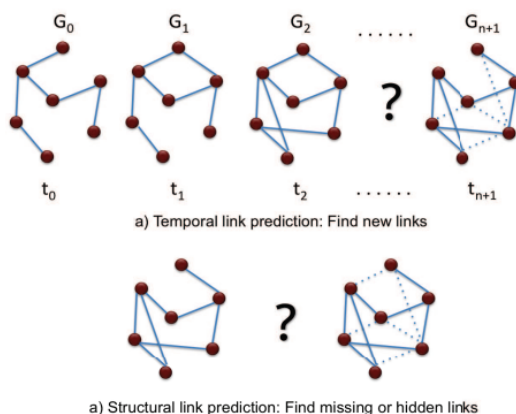
Prediction has long been an integral part of scientific studies since ancient times. For instance, studying the movement of stars and interstellar objects can help researchers predict astronomical phenomena such as eclipse, novae etc. Similarly, by studying the topology of the network and node's attributes, we can predict the missing links or the links that are going to be added in the future. Many social, biological and information systems can be described by networks, where nodes represent entities (individual) and links denote the relationships (interaction) between the nodes. Therefore, the study of complex networks has attracted increasing attention and become a common focus of many branches of science [8]. However, mining and analyzing network data is a non-trivial task as real network data is often incomplete and dynamic. Therefore predicting the missing links in the current network and newly added links in the future is very important for understanding the evolution of the networks and completing the current networks. Link prediction, a task to predict the missing links or new links that will be added to the network in the future has many important applications. It can be used in social networks for new friends recommendation or in e-commerce for recommending products to customers. Link prediction can also be used by security agencies for predicting possible collaboration between criminals [9]. Formally, link prediction can be stated as below [10]:

**Definition 5.1.** In a network  $G = (V, E)$ ,  $V$  is the set of nodes and  $E$  is the set of edges. The link prediction task is to predict whether there is or will be a link  $e(x, y)$  between a pair of nodes  $x$  and  $y$ , where  $x, y \in V$  and  $e(x, y) \in E$ .

Link prediction falls into two categories: *structural* and *temporal* [10, 11]. Figure 5.1 illustrates the two types of link prediction.

**Structural link prediction** refers to the problem of finding missing or hidden links in the networks that may exist but are not directly visible [12]. This type of link prediction can be used to infer missing links from an observed network such as inferring unobserved protein-protein interaction or finding existing criminal links which often remain hidden in a network.

**Temporal link prediction** refers to the problem of finding new links by studying the temporal history of a network [13]. In this type of link prediction, we usually have information about the network till time  $t$  and our goal will be predicting new links that may appear at some point of time in the future say  $t+k$ . This type of link prediction is primarily used in collaboration network- *which researchers are going to collaborate in the future*, social network - *which users will become friends* and e-commerce websites - *which products will the customer buy*.



**Figure 5.1:** Link prediction types

The first type of link prediction uses a static network and make the network complete by adding the predicted possible missing links to the network and the later type works with the network evolution [12]. In this thesis, our aim is to predict the missing links in an observed network so that we can complete the network by adding those missing links to the network which will facilitate our main ambition of inferring hidden facts from the network.

Liben-Nowell et al. [12] demonstrated link prediction by using features intrinsic to the network itself. Their research proved that the network topology indeed contains latent information from which to infer future interaction. Al Hasan et al. [13] explored the use of supervised machine learning methods (Decision Tree, SVM, Naive Bayes) to predict links in co-authorship networks.



In this chapter, we take an unsupervised approach to predict the missing link for different networks from different domains. In the next chapter, we discuss the supervised approaches and compare the two approaches for predicting missing links.

## 5.2 Graph notation and terms

In this section, we briefly explain some of the characteristics and notation of the graph we used throughout the link prediction task. We are given a graph or network  $G = (V, E)$  where  $V$  is the set of *nodes* or *vertices* and  $E$  is the set of *edges* or *links*. The notations are shown in Table 5.1.

Symbol	Meanings
$G$	Graph
$V$	Set of nodes in $G$
$E$	Set of existing edges in $G$
$U$	Set of all possible edges in $G$ (universal set)
$X$	Set of nonexistent links in $G$
$\Gamma(x)$	Set of neighbors of node $x$
$\Gamma_{in}(x)$	Set of incoming neighbors of node $x$
$\Gamma_{out}(x)$	Set of out going neighbors of node $x$
$ \cdot $	Size of the set
$e(x, y)$	An edge between $x$ and $y$ such that $(x, y) \in V$ and $e(x, y) \in E$
$score(x, y)$	Similarity score for an edge $e(x, y)$
$D$	Density of $G$
$Deg_{avg}$	Average degree of $G$
$C$	Cluster coefficient of $G$

**Table 5.1:** Notation and terms

For a directed graph, if self loops are not considered, the size of universal set of edges is  $|U| = |V|(|V| - 1)$  and for an undirected graph,  $|U| = \frac{|V|(|V|-1)}{2}$ . The density of a directed graph is  $D = \frac{2|E|}{|U|}$  and for undirected graph,  $D = \frac{|E|}{|U|}$ . The set of nonexistent links is  $X = U - E$ .

## 5.3 Data Preparation

For this experiment, LOOPS provides us a dataset containing information of the oils and gas related fields. The dataset initially had 5042 nodes and 4504 edges. There are 13 different types of nodes such as Country, License, Well etc. The edges connect two different types of nodes. The dataset can be represented by a directed unweighted graph  $G = (V, E)$  where  $V$  is the set of nodes and  $E$  is the set of directed edges. In the original dataset, a vast majority of the nodes are isolated and only 1773 nodes are connected.

For our link prediction task, we removed all isolated nodes as the algorithms we used for link prediction cannot predict link for isolated nodes, and hence our final network has 1773 nodes and 4504 edges.

For experimental purpose, we use three more real-world datasets. Two of the datasets come from Koblenz Network Collection. These are *Jazz musicians network dataset* [14] where each node represents a Jazz musician and the edge denotes that two musicians have played together in a band and *Hamsterster friendship dataset* [15] where each node is a user of *hamsterster.com* website and the edges represent the friendship. The other dataset is *USAir dataset* [16] where each node represents an airport and the edges represent the airlines. All these three datasets are represented with an unweighted and undirected graph. A summary of the basic topological features of networks is given in Table 5.2.

Nets	V	E	U	D	$Deg_{avg}$	Type
Loops	1773	4504	3141756	0.0029	5.08	Directed
Jazz	198	2742	19503	0.141	27.70	Undirected
Hamsterster	1858	12534	1725153	0.0073	13.50	Undirected
USAir	332	2126	54946	0.0387	12.81	Undirected

**Table 5.2:** The basic topological features of the networks.

## 5.4 Experimental Setup and Evaluation Metric

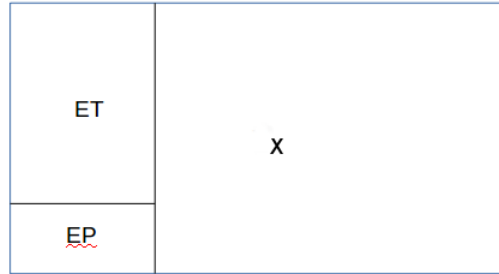
For the missing link prediction, our task is to estimate the existence tendency of all non-observed links based on network topology and nodes attributes. Consider, we are given a graph  $G = (V, E)$  where  $V$  is the set of  $|V|$  nodes and  $E$  is the set of  $|E|$  edges. As we described above, our universal set of links is  $U$  and the set of nonexistent links is  $X = U - E$ . If there are missing links in the network, they must be in  $X$  and our task is to find out these missing links.

To find out the missing links, we use heuristic algorithms. Each algorithm takes an edge  $e(x,y)$  from the set of  $X$  as input and calculates a similarity score  $score(x,y)$ . If we calculate the scores for each nonexistent links and order them by their score in descending order, we can consider that the top-k highest score links are more likely to exist.

Generally, we do not know which links are missing or future links. So we can not blindly trust the algorithms that return the top-k highest score links as missing links. To build trust in our algorithms, we need to test the accuracy of the algorithms first. It is a well-established rule for training and learning algorithms that the existing dataset is divided into two parts: *training set* and *testing set*. The training set is treated as the

known information and the algorithm is trained using the training set. After training the algorithm, the algorithm is used to predict the result of the testing set. Since we know the actual result of the test set, we can compare the result predicted by the algorithm with the actual result using some well-known evaluation metrics.

For the link prediction, our dataset is a graph  $G = (V, E)$ . To test the accuracy of the algorithms, we divide the set of  $E$  into two parts: training set  $E^T$  and test or probe set  $E^P$ . We build a new graph  $G_t = (V, E^T)$  and use our algorithms to predict the missing links in  $G_t$ . Here, one has to notice that the set of  $E^P$  is a subset of nonexistent links  $X$  for  $G_t$ . When the algorithm returns the top scoring missing links, we can check whether links from  $E^P$  appears in the top scoring links. The more the links from  $E^P$  are in top scoring links, the more the accuracy of the algorithms is. Figure 5.2 visualizes the representation of the dataset in details.



**Figure 5.2:** Universal set of edges where ET is training set, EP is test set and X is nonexistent links.

From the Figure 5.2, the following rules hold for the dataset.

$$U = E^T \cup E^P \cup X$$

$$E = E^T \cup E^P$$

$$E^T \cap E^P = \emptyset$$

$$E^T \cap X = \emptyset$$

$$E^P \cap X = \emptyset$$

Dividing dataset into training and testing sets can cause the isolation of some nodes in the network which in turn will give undesired results as the link prediction methods we selected cannot predict unconnected nodes. Hence, we need to be extra careful while splitting the edges into training and test set so that none of the nodes in the network become isolated. We make sure that the pair of nodes  $x, y$  in each  $e(x, y)$  of in  $E^P$  at least has a degree of two. In all network, the training set  $E^T$  constitutes 90% of the

edges of  $E$  while test set  $E^P$  contains the remaining 10% of the edges. The statistic of the training, testing, and nonexistent links are shown in Table 5.3.

Nets	$ E $	$ E^T $	$ E^P $	$ X $
Loops	4504	4054	450	3137702
Jazz	2742	2468	274	17035
Hamsterster	12534	11281	1253	1713872
USAir	2126	1914	212	53032

**Table 5.3:** Statistics of training, testing and nonexistent links

The prediction quality is evaluated by two standard metrics. The first one is called the area under the receiver operating characteristic curve (AUC). The metric gives the probability that a randomly chosen link from  $E^P$  gives a higher score than a randomly chosen link from  $X$  (nonexistent links). If we run the algorithms for  $n$  independent comparisons, say  $n'$  occurrences of links from  $E^P$  gives a higher score and  $n''$  occurrences of links from  $E^P$  and  $X$  have the same score, then we define the accuracy as:

$$AUC = \frac{(n' + 0.5 * n'')}{n}$$

If all the score are generated from an independent and identical distribution, the accuracy should be 0.5. Therefore, the degree to which the accuracy exceeds 0.5 indicates how much better the algorithm performs than pure chance [8]. The second metric we used is called *Precision*, which is a well-known metric widely used in information retrieval. Precision is defined as the fraction of retrieved documents that are relevant. In other word, if we take top  $K$  links as predicted links, among which  $K_r$  links are right (i.e, there are  $K_r$  links in the test set  $E^P$ ), then the precision is  $P = \frac{K_r}{K}$  [17].

## 5.5 Link Prediction Methods

In this section, we discuss five commonly used link prediction methods that use network topology in the prediction process. All the methods described here assign a connection weight  $score(x,y)$  to pair of nodes  $(x,y)$  based on the input graph  $G$ . We can view the  $score(x,y)$  as a measurement of proximity or similarity between node  $x$  and node  $y$  relative to the network topology.

### 5.5.1 Common Neighbors

This method is based on assumption that two nodes  $x$  and  $y$  with a large overlap in their neighbor set of  $\Gamma(x)$  and  $\Gamma(y)$  will be connected in the future. In a social network, common neighbors is a very strong feature for predicting future link between two users. Two users with a lot of mutual friends are more likely to be friend in the future. The score for common neighbors is calculated as:

$$score(x, y) = \begin{cases} |\Gamma(x) \cap \Gamma(y)| & \text{if } G \text{ is undirected} \\ |\Gamma_{out}(x) \cap \Gamma_{in}(y)| & \text{if } G \text{ is directed} \end{cases}$$

### 5.5.2 Jaccard's Coefficient

Jaccard's coefficient or Jaccard index is a well-known similarity metric commonly used in information retrieval. Let  $F$  be a set of features that are either in  $x$  or  $y$ . If we randomly pick a feature  $f$  from  $F$ , what is the probability that both  $x$  and  $y$  have the feature  $f$ . If we take the features here to be neighbors in graph  $G$ , we can measure how likely is a neighbor of  $x$  is to be a neighbor of  $y$  and vice versa. We can calculate the *score* ( $x, y$ ) as Jaccard's Coefficient as:

$$score(x, y) = \begin{cases} \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|} & \text{if } G \text{ is undirected} \\ \frac{|\Gamma_{out}(x) \cap \Gamma_{in}(y)|}{|\Gamma_{out}(x) \cup \Gamma_{in}(y)|} & \text{if } G \text{ is directed} \end{cases}$$

### 5.5.3 Adamic/Adar Index

Adamic and Adar [18] consider a measure to see if two personal home pages are strongly related. To do this, they compute features of the pages and define the similarity between two pages to be

$$similarity(x, y) = \sum_{shareditems} \frac{1}{\log[frequency(shareditems)]}$$

The shared items that are unique to fewer users are weighted more than commonly occurring items. If we consider the shared items as common neighbors, we can define the adamic/adar index measure for link prediction as, if a common neighbor of  $x$  and  $y$  has more neighbors, then it is less likely that he will introduce  $x$  and  $y$  with each other then in case he has only a few friends. The score for adamic/adar is calculated as:

$$score(x, y) = \begin{cases} \sum_{z \in (\Gamma(x) \cap \Gamma(y))} \frac{1}{\log(|\Gamma(z)|)} & \text{if } G \text{ is undirected} \\ \sum_{z \in (\Gamma_{in}(x) \cap \Gamma_{out}(y))} \frac{1}{\log(|\Gamma_{out}(z)| + \epsilon)} & \text{if } G \text{ is directed} \end{cases}$$

where  $\epsilon$  is a very small number ( $\epsilon \ll 1$ ) to avoid denominator to be zero.

### 5.5.4 Preferential Attachment

Work by Barabasi et al [19] on collaboration networks suggests that there is a positive correlation between the probability of collaboration between two nodes  $x, y$  and the product of the number of their neighbors. It is also one of the well-known concepts in social networks that users with more friends tend to create more connection in the future. This corresponds to the measure:

$$score(x, y) = \begin{cases} |\Gamma(x)| * |\Gamma(y)| & \text{if } G \text{ is undirected} \\ |\Gamma_{out}(x)| * |\Gamma_{in}(y)| & \text{if } G \text{ is directed} \end{cases}$$

### 5.5.5 Resource Allocation

Motivated by transferring resource between two unconnected nodes with their common neighbors playing the role of transmitters, T. Zhou et al. [20] proposed a similarity measure as:

$$score(x, y) = \begin{cases} \sum_{z \in (\Gamma(x) \cap \Gamma(y))} \frac{1}{|\Gamma(z)|} & \text{if } G \text{ is undirected} \\ \sum_{z \in (\Gamma_{in}(x) \cap \Gamma_{out}(y))} \frac{1}{|\Gamma_{out}(z)|} & \text{if } G \text{ is directed} \end{cases}$$

## 5.6 Result and Discussion

In this section, we present the result of our experimental evaluation. The prediction accuracy measured by precision and AUC is shown in Table 5.4 and Table 5.5 respectively. For calculating the precision, we set  $K = |E^P|$ , which means the number of retrieved elements equals the number of relevant elements. Under this specific choice of  $K$ , precision is equal to another metric call recall. Recall is defined as the fraction of relevant elements that have been retrieved over the total amount of relevant elements.

In term of precision, for Loops and Jaaz, *adamic/adar* score highest with the accuracy of **0.760** and **0.755** respectively. On the other hand, *Common Neighbor* yields highest accuracy in term of precision for Hamster dataset while for USAir, *Resource Allocation* gain the accuracy of **0.425**. In term of AUC, *Preferential Attachment* gives highest accuracy for both Loops and Jazz datasets. *Jaccard coefficient* and *Resource Allocation* achieve highest accuracy for Hamster and USAir with the accuracy of **0.883** and **0.969** respectively.

Nets	CN	JC	AA	PA	RA
Loops	0.127	0.016	<b>0.131</b>	0.029	0.120
Jazz	0.515	0.496	<b>0.522</b>	0.117	0.518
Hamsterster	<b>0.035</b>	0.029	0.032	0.029	0.030
USAir	0.335	0.066	0.354	0.311	<b>0.425</b>

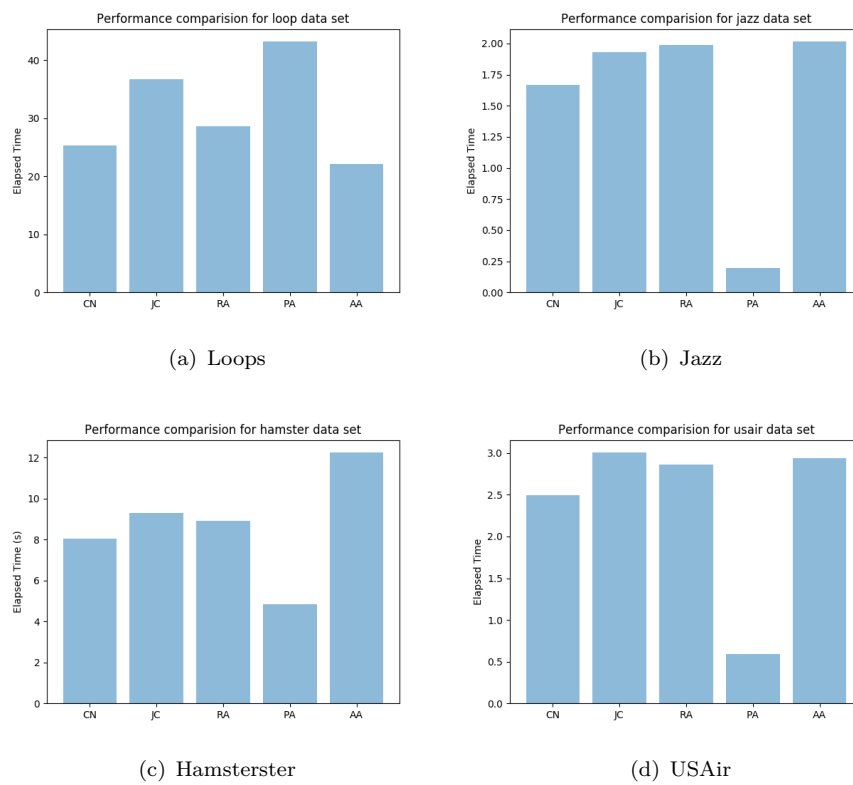
**Table 5.4:** Prediction accuracy measured by Precision

Nets	CN	JC	AA	PA	RA
Loops	0.688	0.687	0.688	<b>0.760</b>	0.689
Jazz	0.931	0.969	0.958	<b>0.755</b>	0.969
Hamsterster	0.809	<b>0.883</b>	0.815	0.870	0.815
USAir	0.962	0.932	0.967	0.939	<b>0.969</b>

**Table 5.5:** Prediction accuracy measured by AUC

Finally, we measure the execution time for each algorithm that we use for link prediction. The elapsed time for each algorithm is given in Figure 5.3. Surprisingly, for Jazz, Hamsterster and USAir datasets which are undirected graphs, *adamic/adar* takes the longest time while *Preferential Attachment* takes the least amount of time to execute. For the Loops dataset which is a directed graph, a totally opposite phenomenon is observed: *adamic/adar* taking least amount of time while *preferential attachment* consumed most of the time for calculating scores for missing links.

Most of the previous link prediction researches were carried out on collaboration or social networks where the graph evolves along with the time. In a time series dataset, the training and test sets are split based on the snapshot of a time. Since none of our datasets comes with time evolution, we have to split the training and test sets randomly. Randomly splitting the edges can have the negative impact on calculating the accuracy of the algorithms as splitting in such way can change the topology of the original network. Taking this into mind, we can assume that if the algorithms are applied on an actual unobserved set of links (without splitting edges), the algorithms would perform better. Moreover, from the result, we can clearly see that the algorithms for all dataset performed well above the random predictor. So we can conclude that the network topology indeed contains the crucial ingredient to predict missing or future links.



**Figure 5.3:** Comparison of execution time for different algorithms on different datasets



## Chapter 6

# Link Prediction - Supervised Approach

In the previous chapter (Chapter-5), we discussed what link prediction is and we demonstrated link prediction using unsupervised approach. In this chapter, we use supervised machine learning approach to predict the missing links in a network. Though we compile supervised learning approach into a separate chapter, we will be using the same networks and the terms that we used in the previous chapter. Hence, readers are advised to read the previous chapter before proceeding.

### 6.1 Supervised Learning

Before diving into the problem, we need to understand what is supervised and unsupervised learnings and how they differ from each other. In machine learning, learning algorithms are typically characterized as *supervised* and *unsupervised*. In supervised learning problem, we start with a data set containing training examples with associate correct *labels* which is often called as *training set*. Once the model is learned using the training set, it can be applied to a set of unlabeled items, called *test set*, in order to automatically apply labels. For example, if we are given a set of emails that have been labeled as *spam* or *not spam*, a classification model can be learned using the given set of emails. Then we can apply the model to classify if an incoming email is *spam* or *not spam*.

Unsupervised learning algorithms, on the other hand, do not have labeled data. For instance, the link prediction we implemented in the previous chapter falls under the

category of an unsupervised solution since no labeled training set is adapted to derive a prediction model.

## 6.2 Limitations of Unsupervised Link Prediction

The unsupervised link prediction presents some limitations. In unsupervised approach, the algorithm ranks the pair of non-connected nodes according to their score and we choose top  $\mathbf{K}$  ranked pairs of nodes as predicted links. However, what should the value of  $\mathbf{K}$  be is still an open question with no concrete answer. Another problem with the unsupervised approach is that the algorithm that is used for ranking of node pairs uses only one metric (for example, *common neighbors*). Using a single metric may not completely explore different structural patterns contained in the network. Considering the above limitations, many researchers adopted the link prediction problem as a supervised machine learning problem.

## 6.3 Dataset and Evaluation metrics

For supervised learning approach, we use the same datasets that we used for link prediction in unsupervised approach. A summary of the dataset is given in Table 5.2. The notation and the terms we often use for the network can be found in Table 5.1.

For evaluating the classification model, we adopted five different performance metrics. *Precision*, *Recall*, *F1-measure*, *AUC Score*, and *Accuracy*. With the help of the *confusion matrix* shown in Table 6.1, we describe how each metric works.

	Predicted: YES	Predicted: NO
Actual: YES	TP	FN
Actual: NO	FP	TN

**Table 6.1:** The confusion matrix

**Accuracy:** Accuracy can be defined as the number of correctly predicted labels in the test network out of the total number of possible examples of unobserved links.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

**Precision:** Precision can be defined as the fraction of correctly predicted links out of total prediction made.

$$Precision = \frac{TP}{TP + FP}$$

**Recall:** Recall is defined as the fraction of correctly predicted links out of the total number of actual new links.

$$Recall = \frac{TP}{TP + FN}$$

**F1-measure:** F1-measure is defined as the harmonic mean of precision and recall.

$$F1 - measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

**Area Under ROC curve (AUC):** AUC is the probability that a randomly chosen positive example ranks above a randomly chosen negative example.

## 6.4 Feature Selection

A key part of many machine learning algorithm is feature selection. In link prediction, a combination of topological and non-topological features are used depending on the availability of such features in the network of consideration. For instance, Al Hasan et al. [13] consider features that are both intrinsic to the network (sum of common neighbors, shortest path, clustering index etc) and extrinsic to the network ( sum of keywords, sum of paper, keyword match count etc) and got promising results with accuracy more than 85%.

Unfortunately, in our dataset, there are no extrinsic features. So, all the features we use for supervised link prediction are intrinsic features - *topological features*. Each link prediction method we used in unsupervised link prediction is used as a feature for supervised link prediction. The features are listed below.

- Common Neighbors (5.5.1)
- Jaccard's Coefficient (5.5.2)
- Adamic/Adar Index (5.5.3)
- Preferential Attachment (5.5.4)
- Resource Allocation (5.5.5)

- Shortest Path

The shortest path is the path between node  $x$  and  $y$  where  $(x, y) \in V$  and  $e(x, y) \in X$ .

For the Loops dataset which is a directed network, we use two more features. These are:

- Common In Neighbors
- Common Out Neighbors

Common In Neighbors is:

$$score(x, y) = |\Gamma_{in}(x) \cap \Gamma_{in}(y)|$$

Common Out Neighbors is:

$$score(x, y) = |\Gamma_{out}(x) \cap \Gamma_{out}(y)|$$

Features normalization is an important preprocessing step in many machine learning algorithms. So we normalize the features such that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one before using them in the classification model.

## 6.5 Class imbalance and under-sampling

In a typical binary classification problem, positive and negative classes are approximately balanced. As a result, we can calculate expectation for a baseline classifier performance. For example, if the classes are balanced, we can assume that a classifier that predicts the label randomly or a classifier that assigns all positive or all negative classes will have a performance of 0.5 for all the evaluation metrics mentioned above.

However, binary classification problems that exhibit class imbalance do not share this property [10]. Link prediction domain is an extreme example of class imbalance. The number of present links (positive class) in a network is magnitude smaller than the number of absent link (negative class) in the network. For example, in our **Loops** dataset, the number of present links is **4504** while there are over **3 millions** absent links.

Accuracy metrics is biased toward class imbalance. For instance, if the ratio of positive and negative class distribution is 10:90, and a classifier algorithm that just classifies all

instances to be negative will have the accuracy of 90%. Even though the accuracy is very high, this type of classifiers are useless as they do not predict any of the positive instances. Also, during the training period of the algorithm, due to extreme class imbalance, the information carried by positive class get diluted in the vast negative class. Moreover, unlike classical classification problem where overall prediction accuracy is important, in link prediction, classification of positive examples are more important [11].

A straightforward remedy to class imbalance is *under-sampling* or *down-sampling* of majority class. In under-sampling, some of the instances from majority class are randomly removed from the training set. M. Kubat et al. [21] proposed to selectively under-sample the majority class while keeping all minority class. A careless under-sampling may lose valuable information and so, a careful selection should be made on the criteria deciding which examples are to be discarded from the training sets. In our case, we keep all the positive instances and take the same number of negative instances that are within three hops distance.

## 6.6 Link Prediction using Supervised Learning

After the work of Liben-Nowell et al. [12], who demonstrated that simple topological features representing relationships between pairs of non-connected nodes in a complex network can be used for predicting new forming links, many attempts were made to combine the effects of individual topological features in order to enhance the overall prediction performance of the approach. Most of these works use machine learning algorithms.

The first approach we studied is proposed by Al Hasan et al. [13] who converted the link prediction problem into a binary classification problem where the examples are non-connected pairs of nodes in the network. They use different supervised learning methods such as SVM, Decision Tree, and Naive Bayes to predict links in co-authorship networks obtained from BIOBASE and DBLP datasets. They partition the network into two non-overlapping graphs  $G_1$  and  $G_2$  based on the publication years.  $G_1$  is used as the training years. They calculate features from  $G_1$  for all nonexistent links and each such link is assigned a positive or negative label depending on whether that nonexistent link in  $G_1$  is formed in  $G_2$ . Having a set of nodes with features and label, one can construct any classification algorithm to generate a model which can further be used to classify test instances with the same feature vector.

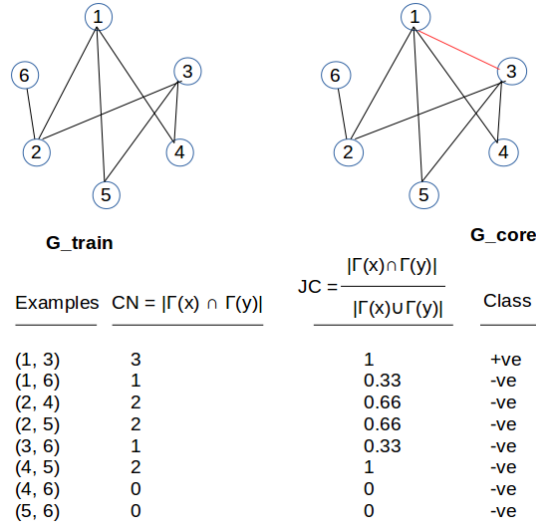
Taking the motivation from Al Hasan et al. [13], we devised a model to predict the missing links in a network using supervised learning. Consider, we are given a network

$G_{core} = (V, E)$  and the set of absent links or nonexistent links in  $G_{core}$  is denoted by  $X$ . Our task is to identify if there are any potential missing links in the set of absent links  $X$ .

To predict the missing links, we use the following rules:

- Let  $E''$  hold 10% of the random edges from  $G_{core}$ .
- We build a train graph  $G_{train} = (V, E - E'')$ . In other word,  $G_{train}$  is build upon the original graph by randomly removing 10% of the edges.
- For each non existent links  $e(x, y)$  in  $G_{train}$  i.e  $e(x, y) \notin (E - E'')$ , an example is build using the following information.
  - A feature set of topological attributes for  $e(x, y)$  is computed. In section 6.4, we detail a list of topological features.
  - A class label is associated to the example: if the link  $e(x, y)$  exist in  $G_{core}$ ,  $e(x, y)$  is a positive example. It is a negative example otherwise.

A graphical representation of features generation and label assignment for training data is shown in Figure 6.1. In the figure, only two features are considered.



**Figure 6.1:** Generation of training set on sample graph

The statistic of training set before and after under-sampling is shown in Table 6.2.

Since the examples are extremely imbalance, we perform under-sampling as discussed in Section 6.5. We keep all the positive instances and select an equal number of negative instances from the pool of negative examples that are within three hops distance. We then normalize the features such that the features have a mean of zero and standard deviation of one.

Nets	Train ( <i>Before sampling</i> )		Train ( <i>After sampling</i> )	
	+ve	-ve	+ve	-ve
Loops	450	3137252	450	450
USAir	212	52820	212	212
Jazz	274	16761	274	274
Hamster	1253	1712619	1253	1253

**Table 6.2:** Number of positive and negative classes in different datasets

Finally, a supervised learning technique is applied to compute a predictive model that discriminates positive examples from negative ones.

Traditionally, in machine learning, the models are evaluated on examples which have not been seen by the system while learning. One of the most used ways to evaluate classification models in machine learning is *K-fold* cross-validation. In *K-fold* cross-validation, the training dataset is divided into *K* subsets. The validation is performed for *K* times. Each time, one of the *K* subset is used as the test set and the other *K-1* subsets are put together to form a training set. The final result is the average of all trials. *K-fold* cross-validation significantly reduces model over-fitting and under-fitting as all the subsets are being used both as training set and test set. In this research, we use 10-fold cross validation for the result reported. The classification algorithms we used for this research are discussed in next section.

## 6.7 Classification Algorithms

For supervised learning, there exist so many algorithms. Depending on the domain and dataset, different algorithm performs differently. In this research, we experimented with three different classification algorithms. The algorithms that we used are *Decision Tree*, *Random Forest* and *SVM with linear kernel*. For all the classification algorithms, we use *Scikit-Learn*, a machine learning library for *Python* programming language.

Then we compare the performance of the above classifiers using different performance metrics such as *accuracy*, *precision*, *recall* etc. More details of the performance metrics are discussed in Section 6.3. For all the algorithms, we used 10-fold cross validation for the result reported.

## 6.8 Result and Discussion

Table 6.3, 6.4, 6.5 and 6.6 shows the performance comparison for different classifiers on Loops, USAir, Jazz and Hamster datasets respectively. For Loops, USAir and Jazz,

all the classifier exceed the accuracy above 80%. For the Hamster, the accuracy for all models reached above 70%. From the result, we can see that *Random forest* classifier gives highest accuracy for *Loops* and *Jazz* datasets whereas *Decision Tree* performs better in *USAir* and *Hamster* datasets.

Comparing the result with the unsupervised approach, we can see a huge improvement in all the datasets. Since the accuracy and all other performance measures exceed the baseline classifier which could be around 50%, we can conclude that the features we selected have good discrimination ability. In unsupervised link prediction, each method uses a single topological attribute for calculating the score. In supervised learning, however, we combine different topological attributes into the set of a feature vector. As a result, we can assume that link prediction using supervised learning achieve good result compare to unsupervised approach.

Loops Dataset					
Classifier	Accuracy	Precision	Recall	F1-measure	AUC
SVM	0.800	0.817	0.769	0.774	0.800
<b>Random Forest</b>	0.851	0.848	0.856	0.846	0.851
Decision Tree	0.822	0.819	0.829	0.817	0.822

**Table 6.3:** Performance of different classification algorithms for Loops dataset

USAir Dataset					
Classifier	Accuracy	Precision	Recall	F1-measure	AUC
SVM	0.856	0.933	0.768	0.836	0.856
Random Forest	0.877	0.893	0.858	0.871	0.877
<b>Decision Tree</b>	0.889	0.917	0.858	0.881	0.889

**Table 6.4:** Performance of different classification algorithms for USAir dataset

Jazz Dataset					
Classifier	Accuracy	Precision	Recall	F1-measure	AUC
SVM	0.900	0.927	0.873	0.896	0.900
<b>Random Forest</b>	0.924	0.935	0.913	0.922	0.924
Decision Tree	0.855	0.857	0.859	0.852	0.855

**Table 6.5:** Performance of different classification algorithms for Jazz dataset

Hamster Dataset					
Classifier	Accuracy	Precision	Recall	F1-measure	AUC
SVM	0.753	0.777	0.709	0.741	0.753
Random Forest	0.748	0.799	0.664	0.723	0.748
<b>Decision Tree</b>	0.759	0.785	0.712	0.746	0.759

**Table 6.6:** Performance of different classification algorithms for Hamster dataset



## Chapter 7

# Conclusion and Further Work

This chapter concludes this thesis and presents suggestions for further work.

### 7.1 Conclusion

For this thesis, **LOOPS** provides us the required dataset. In the dataset, we notice that there exists so many redundant, conflicting and inconsistent data. Therefore, we devise a system called **object extractor** that extracts only relevant information from the raw dataset. The information we extract is refurbished so that the data are consistent. We effectively handle data ambiguity and data redundancy for a better representation of data with very little to no noise. Due to the large size of the data, we have to come up with an effective data structure that can manipulate the large volume of data within a reasonable time.

The *object extractor* system basically extracts object and their attributes and the relationships between different objects. To represent the extracted information, we develop **ger**, a system that represents the extracted data using a graphical data model. By using *ger system*, we demonstrate that if we store the data as graph, we can recognize the entities in a file using the graph and infer more knowledge about the recognized data using graph traversal techniques. We also demonstrate that, for the linked data, it is possible to use graph algorithms for hierarchical inference.

In the network generated by *ger* system, we notice that some of the links are missing. Therefore, we are interested if we can predict the missing links in the network. We analyze the topology of the graph and implement link prediction algorithms to add the potential missing links to the graph. We use both supervised and unsupervised link prediction methods and we find that the supervised link prediction method can predict

missing link better than it's counterpart. We notice that the topology of the graph carries vital information that can be used to predict missing link in a network.

We initially thought that, if we use link prediction and add potential missing links to the graph, the graph will give a better result for inferencing. However, to our surprise, there was no change in the result for inferencing. This is because the inferencing we use in this work is hierarchical inference. In a hierarchical inference, if B is subclass of A and C is subclass of B, then we can infer that C is also subclass of A because there is a path from A to C via B. Adding a direct path from A to C will make no difference for reference as long as there is a via path from A to C. Link prediction predicts and adds the missing links such as a direct link from A to C. The benefit of the link prediction would be apparent if we can use link prediction and add links to the nodes that are isolated in the original graph. Unfortunately, the link prediction algorithms we use do not support isolated nodes as we do not have enough features except the topological features. For isolated nodes, there are no topological features, hence the isolated nodes are not considered for link prediction in this work.

Although link prediction does not have much impact on the domain of this work, we gain a considerable knowledge from studying the topology of the data. Networks such as social and collaborative networks show a great demand for link prediction. Therefore, we believe that the effort we put for link prediction in this thesis is a good effort that we can utilize in other domain of networks.

Finally, we firmly believe that the linked data representation with graph and analyzing the graph gives a deep insight of the data that will facilitate in taking further action.

## 7.2 Further Work

This section provides some thoughts on further improvement of the work presented in this thesis.

**Storing data in graph database:** The data we use for building graph is stored in memory. Therefore, every time we use the algorithm, we need to load data and build the graph. Storing the graph in a graph database will eliminate the graph building process on every use.

**A more intelligent Entity Recognition Algorithm:** The entity recognition algorithm we use in this thesis is a simple and generic substring matching algorithm. For example, if we have a node "D12 E" in the graph and a text "D12 Enable" is given as input, the algorithm would match "D12 E" for the text "D12 Enable ". If we do not want

to have such behavior, a more complex algorithm that understands the formatting of the text should be implemented.

**Link prediction for isolated nodes:** The algorithms used for link prediction do not support isolated nodes. We believe, an algorithm that can predict missing link for isolated nodes will make the graph more connected.



# List of Figures

2.1	Representation social graph . . . . .	9
3.1	A sample Excel sheet with data to be extracted (Type I) . . . . .	14
3.2	A sample Excel sheet with data to be extracted (Type II) . . . . .	14
3.3	Illustration of objects extraction . . . . .	17
3.4	Illustration of relationships extraction . . . . .	19
3.5	Structure of object sheet . . . . .	22
3.6	Structure of relationship sheet . . . . .	22
4.1	Different type of graphs . . . . .	27
4.2	Representation of graph . . . . .	27
4.3	Visualization of LOOPS data in a graph . . . . .	30
4.4	Example of Inference . . . . .	33
4.5	Sample dataset for entity recognition and inferencing . . . . .	35
4.6	Sample of recognized and inferred entities . . . . .	36
5.1	Link prediction types . . . . .	38
5.2	Universal set of edges where ET is training set, EP is test set and X is nonexistent links. . . . .	41
5.3	Comparison of execution time for different algorithms on different datasets	46
6.1	Generation of training set on sample graph . . . . .	52



# List of Tables

3.1	object extraction by object extractor . . . . .	22
4.1	Result of recognized and inferred entities for sample dataset . . . . .	36
5.1	Notation and terms . . . . .	39
5.2	The basic topological features of the networks. . . . .	40
5.3	Statistics of training, testing and nonexistent links . . . . .	42
5.4	Prediction accuracy measured by Precision . . . . .	45
5.5	Prediction accuracy measured by AUC . . . . .	45
6.1	The confusion matrix . . . . .	48
6.2	Number of positive and negative classes in different datasets . . . . .	53
6.3	Performance of different classification algorithms for Loops dataset . . . . .	54
6.4	Performance of different classification algorithms for USAir dataset . . . . .	54
6.5	Performance of different classification algorithms for Jazz dataset . . . . .	54
6.6	Performance of different classification algorithms for Hamster dataset . . . . .	54





# Listings

3.1	Header extraction . . . . .	16
3.2	Data normalization . . . . .	18
4.1	Representation of node in Java . . . . .	28
4.2	Pseudocode for building graph . . . . .	29
4.3	Pseudocode for Entity Recognition in a graph . . . . .	31
4.4	Algorithm for Inference in a graph . . . . .	34
A.1	Feature score calculation for supervised link prediction . . . . .	65
A.2	Decision Tree algorithm for supervised link prediction . . . . .	66
A.3	Random Forest algorithm for supervised link prediction . . . . .	67
A.4	SVM algorithm for supervised link prediction . . . . .	69



## Appendix A

# Appendix

```
1 import networkx as nx
2 from math import log
3 import sys
4
5
6 def base_common_neighbors(G, u, v):
7     if G.is_directed():
8         return (w for w in G.successors(u) if w in G.predecessors(v) and w not in (u, v))
9     return nx.common_neighbors(G, u, v)
10
11
12 def common_neighbors(G, u, v):
13     return len(list(base_common_neighbors(G, u, v)))
14
15
16 def common_neighbors_in(G, u, v):
17     return len(list((w for w in G.predecessors(u) if w in G.predecessors(v) and w not in (u, v))))
18
19
20 def common_neighbors_out(G, u, v):
21     return len(list((w for w in G.successors(u) if w in G.successors(v) and w not in (u, v))))
22
23
24 def jaccard_coefficient(G, u, v):
25     if G.is_directed():
26         union_size = len(set(G.successors(u)) | set(G.predecessors(v)))
27         if union_size == 0:
28             return 0
29         return len(list(base_common_neighbors(G, u, v))) / union_size
30     else:
31         jc = list(nx.jaccard_coefficient(G, [(u, v)]))
32         return jc[0][2]
```

```

33
34
35 def adamic_adar(G, u, v, epsilon=0.001):
36     if G.is_directed():
37         return sum(1 / log(G.out_degree(w) + epsilon) for w in base_common_neighbors(G, u, v))
38     else:
39         aa = list(nx.adamic_adar_index(G, [(u, v)]))
40         return aa[0][2]
41
42
43 def resource_allocation(G, u, v):
44     if G.is_directed():
45         return sum(1 / G.out_degree(w) for w in base_common_neighbors(G, u, v))
46     else:
47         ra = list(nx.resource_allocation_index(G, [(u, v)]))
48         return ra[0][2]
49
50
51 def preferential_attachment(G, u, v):
52     if G.is_directed():
53         return G.out_degree(u) * G.in_degree(v)
54     else:
55         pa = list(nx.preferential_attachment(G, [(u, v)]))
56         return pa[0][2]
57
58
59 def shortest_distance(G, u, v):
60     try:
61         sp = nx.shortest_path_length(G, u, v)
62     except nx.NetworkXNoPath:
63         sp = sys.maxsize
64     return sp

```

**Listing A.1:** Feature score calculation for supervised link prediction

```

1 from sklearn.preprocessing import StandardScaler
2 import numpy as np
3 from sklearn.tree import DecisionTreeClassifier
4 import tools
5 from sklearn.model_selection import StratifiedKFold
6 import sklearn.metrics as metrics
7
8
9 def decision_tree_model(train_file):
10     X_train, Y_train = tools.load_features(train_file)
11
12     # normalize data
13     scaler = StandardScaler()

```

```

14
15 X_train = scaler.fit_transform(X_train)
16
17 # tuned hyperparameters
18 parameters = {
19     "criterion": "entropy", # default = gini
20     "max_depth": 5, # 9
21     "min_sample_leaf": 4
22 }
23 # instantiate classifier
24 decision_tree_clf = DecisionTreeClassifier(criterion=parameters["criterion"],
25                                           max_depth=parameters["max_depth"],
26                                           min_samples_leaf=parameters["min_sample_leaf"])
27
28 k = 10
29 kf = StratifiedKFold(n_splits=k)
30 precisions = []
31 recalls = []
32 f1_scores = []
33 accuracies = []
34 auc_roc_scores = []
35
36 print("\n Decision Tree Classifier starting... \n")
37 for train_index, test_index in kf.split(X_train, Y_train):
38     x_train, x_test = X_train[train_index], X_train[test_index]
39     y_train, y_test = Y_train[train_index], Y_train[test_index]
40
41     decision_tree_clf.fit(x_train, y_train)
42     y_pred = decision_tree_clf.predict(x_test)
43
44     accuracies.append(metrics.accuracy_score(y_test, y_pred))
45     precisions.append(metrics.precision_score(y_test, y_pred))
46     recalls.append(metrics.recall_score(y_test, y_pred))
47     f1_scores.append(metrics.f1_score(y_test, y_pred))
48     auc_roc_scores.append(metrics.roc_auc_score(y_test, y_pred))
49
50 print("\n==== Report=====\n")
51 print("Accuracy Avg: %.3f \n" % (np.mean(accuracies)))
52 print("Precision Avg: %.3f \n" % (np.mean(precisions)))
53 print("Recall Avg: %.3f \n" % (np.mean(recalls)))
54 print("F1-measure Avg: %.3f \n" % (np.mean(f1_scores)))
55 print("AUC Avg: %.3f \n" % (np.mean(auc_roc_scores)))

```

**Listing A.2:** Decision Tree algorithm for supervised link prediction

```

1 from sklearn.preprocessing import StandardScaler
2 import numpy as np
3 from sklearn.ensemble import RandomForestClassifier

```

```
4 import tools
5 from sklearn.model_selection import StratifiedKFold
6 import sklearn.metrics as metrics
7
8
9 def random_forest_model(train_file):
10     X_train, Y_train = tools.load_features(train_file)
11
12     # normalize data
13     scaler = StandardScaler()
14     X_train = scaler.fit_transform(X_train)
15
16     # tuned hyperparameters
17     parameters = {
18         "n_estimators": 100,
19         "criterion": "entropy", # default = gini
20         "max_depth": 5, # 9
21         "min_samples_leaf": 5, # 10
22         "bootstrap": True,
23         "n_jobs": -1
24     }
25
26     # instantiate classifier
27     rf_clf = RandomForestClassifier(
28         n_estimators=parameters["n_estimators"],
29         criterion=parameters["criterion"],
30         max_depth=parameters["max_depth"],
31         min_samples_leaf=parameters["min_samples_leaf"],
32         bootstrap=parameters["bootstrap"],
33         n_jobs=parameters["n_jobs"]
34     )
35
36     k = 10
37     kf = StratifiedKFold(n_splits=k)
38     precisions = []
39     recalls = []
40     f1_scores = []
41     accuracies = []
42     auc_roc_scores = []
43
44     print("\n Random Forest Classifier starting... \n")
45     for train_index, test_index in kf.split(X_train, Y_train):
46         x_train, x_test = X_train[train_index], X_train[test_index]
47         y_train, y_test = Y_train[train_index], Y_train[test_index]
48
49         rf_clf.fit(x_train, y_train)
50         y_pred = rf_clf.predict(x_test)
51
```

```

52     accuracies.append(metrics.accuracy_score(y_test, y_pred))
53     precisions.append(metrics.precision_score(y_test, y_pred))
54     recalls.append(metrics.recall_score(y_test, y_pred))
55     f1_scores.append(metrics.f1_score(y_test, y_pred))
56     auc_roc_scores.append(metrics.roc_auc_score(y_test, y_pred))
57
58     print("\n==== Report=====\n")
59     print("Accuracy Avg: %.3f \n" % (np.mean(accuracies)))
60     print("Precision Avg: %.3f \n" % (np.mean(precisions)))
61     print("Recall Avg: %.3f \n" % (np.mean(recalls)))
62     print("F1-measure Avg: %.3f \n" % (np.mean(f1_scores)))
63     print("AUC Avg: %.3f \n" % (np.mean(auc_roc_scores)))

```

**Listing A.3:** Random Forest algorithm for supervised link prediction

```

1  from sklearn.preprocessing import StandardScaler
2  import numpy as np
3  from sklearn import svm
4  import tools
5  from sklearn.model_selection import StratifiedKFold
6  import sklearn.metrics as metrics
7  import pandas as pd
8
9
10 def svm_model(train_file):
11     X_train, Y_train = tools.load_features(train_file)
12
13     # normalize data
14     scaler = StandardScaler()
15     X_train = scaler.fit_transform(X_train)
16
17     if testfile is not None:
18         X_test = pd.read_csv(testfile, index_col=None)
19         del X_test["node1"]
20         del X_test["node2"]
21         X_test = scaler.fit_transform(X_test)
22
23     # tuned hyperparameters
24     parameters = {
25         'C': 0.1,
26         'gamma': 0.01,
27         'kernel': "linear"
28     }
29     # instantiate classifier
30     svm_classifier = svm.SVC(C=parameters['C'],
31                             gamma=parameters['gamma'],
32                             kernel=parameters['kernel'],
33                             probability=True)

```

```
34
35 k = 10
36 kf = StratifiedKFold(n_splits=k)
37 precisions = []
38 recalls = []
39 f1_scores = []
40 accuracies = []
41 auc_roc_scores = []
42
43 print("\n SVM Classifier starting... \n")
44 for train_index, test_index in kf.split(X_train, Y_train):
45     x_train, x_test = X_train[train_index], X_train[test_index]
46     y_train, y_test = Y_train[train_index], Y_train[test_index]
47
48     svm_classifier.fit(x_train, y_train)
49     y_pred = svm_classifier.predict(x_test)
50
51     accuracies.append(metrics.accuracy_score(y_test, y_pred))
52     precisions.append(metrics.precision_score(y_test, y_pred))
53     recalls.append(metrics.recall_score(y_test, y_pred))
54     f1_scores.append(metrics.f1_score(y_test, y_pred))
55     auc_roc_scores.append(metrics.roc_auc_score(y_test, y_pred))
56
57 print("\n==== Report=====\n")
58 print("Accuracy Avg: %.3f \n" % (np.mean(accuracies)))
59 print("Precision Avg: %.3f \n" % (np.mean(precisions)))
60 print("Recall Avg: %.3f \n" % (np.mean(recalls)))
61 print("F1-measure Avg: %.3f \n" % (np.mean(f1_scores)))
62 print("AUC Avg: %.3f \n" % (np.mean(auc_roc_scores)))
```

**Listing A.4:** SVM algorithm for supervised link prediction



# Bibliography

- [1] Eric schmidt: Every 2 days we create as much information as we did up to 2003. <https://techcrunch.com/2010/08/04/schmidt-data/>.
- [2] Thomas B. Passin. Explorer’s guide to the semantic web, 2004.
- [3] Yongluan Zhou et al. Semstore: A semantic-preserving distributed rdf triple store, 2014.
- [4] Sebastian Rudolph Pascal Hitzler, Markus Krötzsch. *Foundation of Semantic Web Technologies*. CRC Press, 2010.
- [5] *Graph Databases, 2nd ed.* O’Reilly, 2015. URL <https://pdfs.semanticscholar.org/f511/7084ca43e888fb3e17ab0f0e684cced0f8fd.pdf>.
- [6] Thomas H. Cormen et al. Introduction to algorithms, 3rd ed, .
- [7] Inference. <https://www.vocabulary.com/dictionary/inference>.
- [8] T.Zhou L.Lu. Link prediction in complex network: A survey. <https://arxiv.org/pdf/1010.0725.pdf>, October 2010.
- [9] W. Peng et el. Link prediction in social networks: the state-of-the-art. *Science China - Information Sciences*, January 2015.
- [10] N. V. Chawala Y. Yang, R. N. Lichtenwalter. Evaluating link prediction methods, May 2015.
- [11] Manisha Pujari. *Link Prediction in Large-scale Complex Networks*. PhD dissertation, University of Paris Nord.
- [12] Liben-Nowell et el. The link prediction problem for social networks\*, 2004.
- [13] M. Al Hasan et al. Link prediction using supervised learning \*, .
- [14] Jazz musicians network dataset – KONECT, September 2016. URL <http://konect.uni-koblenz.de/networks/arenas-jazz>.

- 
- [15] Hamsterster friendships network dataset – KONECT, September 2016. URL <http://konect.uni-koblenz.de/networks/petster-friendships-hamster>.
- [16] Vladimir batagelj and andrej mrvar (2006): Pajek datasets. URL <http://vlado.fmf.uni-lj.si/pub/networks/data/>.
- [17] L. Lu C. K. Hu Liming Pan, T. Zhou. Predicting missing links and identifying spurious links via likelihood analysis. *Scientific Report*, 2016.
- [18] Lada A. Adamic and Eytan Adar, 2003.
- [19] Z. NÁt eda E. Ravasz A. Schubert A. L. Barabasi, H. Jeong and T. Vicsek. Evolution of the social network of scientific collaboration.
- [20] Y.-C. Zhang T. Zhou, L. Lu. Predicting missing links via local information. *Eur. Phys. J. B 71 (2009) 623*. URL <http://arxiv.org/pdf/0901.0553.pdf>.
- [21] et al Miroslav Kubat, Stan Matin. Addressng the curse of imbalanced training sets: one-sided selection. URL <http://sci2s.ugr.es/keel/pdf/algorithm/congreso/kubat97addressing.pdf>.