Graph mining for role extraction in predictive analytics of high-performance computing systems

by

Luis Enrique Bobadilla Dias

B.S., Kansas State University, 2018

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2020

Approved by:

Major Professor
William Hsu

Copyright

Abstract

This thesis addresses the task of analyzing property graphs in system log data from high-performance computing (HPC) systems, to identify entity roles to aid in predicting job submission outcomes. This predictive analytics project uses inductive learning on historical logs to produce regression models for estimating resource needs and potential shortfalls, and classification models that predict when jobs will fail due to insufficient resource allocation. The log files are generated by the workload manager of an HPC compute cluster and include runtime parameters for every submitted job. The research objectives of the overall project consist of using these techniques to solve three extant problems: (1) predicting the sufficiency of resource requested in a HPC system at job submission time; (2) making HPC resource allocation more efficient; and (3) building a decision support system for HPC users. Previous approaches and techniques used features such as user demographics and simulations harnessed with simple optimization algorithms to improve the resource allocation usage on a large-scale compute cluster (Kansas State University's Beocat). In this thesis, role extraction is applied with the goal to create a user-specific feature for machine learning tasks. Specific use cases include personalized prediction of submitted job outcomes or reinforcement learning from simulation for optimization tasks in job scheduling. Objectives include improving on the accuracy, precision, recall, and utility of previous learning systems.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my major professor, Dr. William Hsu, for his guidance. I would also like to thank the other members of my committee, Dr. Dan Andresen and Dr. Pascal Hitzler for their guidance and time they spent participating in my committee.

Dedication

This thesis is dedicated to my parents, Norma Sueli Dias de Bobadilla and Marcial Bobadilla Guillen; thank you for your support and encouragement in finishing my degree. I would also like to dedicate this thesis to my girlfriend Diana Emilce Vera Mendez, for her love and patience.

Chapter 1

Introduction

In this chapter I will set the stage for what I aim to accomplish in this thesis.

1.1 Overview

High-performance computing (HPC) interdisciplinary usage is growing rapidly as real-world problems require large data sets to be mined and otherwise analyzed. Many users of HPC systems lack the training to maximize usage and time. The lack of training leads to many submitted jobs to fail, due to not enough resources being allocated or too many resources allocated for a submitted job. There currently exists no globally used guideline for the user to have a way to predict if their job will fail, or even for a recommender system provided by the cluster's workload manager to alert a user when the submitted parameters for a job are insufficient. The input for this desired prediction can be specific to the type of job a user is submitting, or based upon prior information about the user, such as the academic department they belong to or their background and expertise.

The goal of this thesis is to introduce roles as a graph feature to the data set of submitted jobs for an HPC system to improve overall understanding of the behavior between users. Understanding the comparative behavior of Beocat users will help determine a possible social reasoning behind the errors in usage of the cluster.

1.2 Motivation

One of the main motivations for this work stems from two previous published papers from Kansas State University that attempt to solve this same issue. The first of the two published papers ”Machine Learning for Predictive Analytics of Compute Cluster Jobs”, tackled the issue using machine learning, specifically by training a supervised inductive learning system that would predict job failure rates based on user inputted parameters.[1] The second of the published papers, ”Improving HPC System Performance by Predicting Job Resources via Supervised Machine Learning”, took a supervised machine learning model and integrated it with Slurm resource manager simulator to attempt to predict memory requirements for submitted jobs.[2] It is important to note that the data sets for these two papers differ in features and systems. These differences will be discussed in the following chapters. Taking into account the features that were carried over from these experiments, my aim was to add more relevant features to improve measures of goodness for the prediction tasks, such as the precision and accuracy of the classifier.

Some questions I set out to answer in this thesis are:

- How can we best predict failure rates among different users?
- Can we generalize users into different roles?
- Will the inclusion of graph mining features improve or hinder the current precision and accuracy of previous work on predicting failure rate?

Two approaches were already applied by fellow researchers. Machine learning models using user data collected from LDAP and simulation, using a Slurm simulator with the ML model, as input. The approach outlined in this thesis will attempt to improve the user data by assigning roles to users as a means to classify similarities.

Secondly, graph features are often used to improve the performance of a task in a given domain. The motivation behind studying roles as graph features is based on previous work that generalized behavior among nodes. Previous work includes calculating membership of different classes, such as IP communication networks and Bluetooth proximity networks.[3] In each of these data sets, RolX (the role extraction algorithm to be used) assigns roles with better accuracy than a logForest model which uses the same structural information as RolX but doesn't assign roles to the data.[3] The challenge lies in selecting the node features that best represent users. This is because in the given domain, there are many possible links between users and the jobs they submitted.

The novel contribution is the creation of a graph feature which captures similarity in structure across users in an HPC system.

1.3 Problem Statement

High-performance computing (HPC) users whose disciplines provide limited training on HPC itself lose valuable time in their research and development efforts by having to resubmit jobs with different parameters in order to get results. Furthermore if the resources are being used in an inefficient manner on Beocat, the number of users benefiting from the platform is minimized.

Neither of these approaches grant production level predictions to be implemented into Beocat's workload manager as a recommendation system to be sent out to Beocat users. Because of this I continue to attempt different approaches to understanding how to classify users. In exploring ways to represent jobs, I set out to adapt a method for social network analysis (SNA), which is used to study information and learn from users of social media sites (Facebook, Instagram, etc). Although I am not specifically using a social network, I created a graph database with Beocat users as the main entity. This thesis attempts to incorporate graph mining and roles as a relevant feature to add to the existing machine learning algorithms.

1.4 Objectives and Significance of the Thesis

In the following sections I will layout the objectives, goals and significance of this thesis.

1.4.1 Objectives

The main objective for this thesis is to improve upon the work of previous papers of predicting failure rates and improving the resource use on Beocat. One or more constructed features based on roles would be added to a production ready application that can proactively recommend memory size among other parameters to the users before they submit the job to execute. Choosing the correct graph mining techniques and algorithms to develop these features is a crucial step of the research presented in this thesis.

1.4.2 Significance

Part of the significance of this research is the use of graphs to represent the flat data given to us by Beocat's workload managing system, SLURM. This is a novel approach to visualizing and working with the accounting manager's data. Once in a graph format users from the HPC system are able to be examined for structural similarities and roles are assigned in order to better classify membership. The use of machine learning jointly with added features from graphs will yield a greater understanding of different users and can potentially make personalized recommendations based upon the background and experience of said users.

Chapter 2

Background

This chapter presents some background information related to both HPC and graph mining.

2.1 Beocat

At Kansas State University we have a high performance computing cluster called Beocat (https://Beocat.ksu.edu/). Our goal is attempt to have the most efficient system utilization and resource allocation cluster. Beocat users span from all disciplines at the university, from biology, statistics, and a range of engineering departments. When users submit their jobs to Beocat they specify the number of nodes, CPUs, memory, and time limit. This sometimes leads to an issue for less experienced users. They can either underestimate or overestimate some of those parameters leading their job to fail or use up unnecessary system resources. Beocat restricts the maximum time limit to 30 days and number of cores to 200 for a given job.

2.2 Neo4j and property graphs

Property graphs share the same basic representation as most graphs. They are composed of vertices and edges (G = V, E). In property graphs, the terms for these are *nodes* and *relationships*. Along with having nodes and relationships, what makes a property graph unique are *properties*, satellite attributes that can be attached either to the nodes or the relationships.

We now discuss how nodes are defined. They are considered entities. They can hold any amount of data as key-value pairs (properties). Nodes can also be labeled, to specify what domain in the graph they belong to (sub-graphs).

Relationships are similar to edges in a traditional graph except that they have more minimum restrictions in a property graph. Relationships in a property graph must include a start node, end node, direction and a type. Just as nodes have properties, relationships can also hold properties. Moreover, even though relationships have a direction, they can be easily traversed in any direction.

Neo4j is a native graph database that implements the data structure for the property graph mentioned above. *Neo4j* is an open-source, NoSQL database that provides an ACID-compliant back end seen in many other databases. In the next section we will dive into more specifics of Neo4j, and how to use a declarative query language called *Cypher* that is similar in many ways to SQL but is optimized to work with graph databases.

2.3 *Cypher* Query Language

The *Cypher* query language is the query language used in Neo4j to interact with the property graph. (https://neo4j.com/developer/cypher-query-language/)[4]

Cypher shares many features with SQL but with the addition of graph like features. As described above property graphs are mainly comprised of nodes and relationships.

Nodes in *Cypher* are variables that are within parenthesis. Below are some examples of Nodes as described in the documentation.

Cypher is written with readability in mind. Nodes are written as variables within parenthesis. The developers of *Neo4j* chose variables that describe nodes to be in circles as they visually correspond with the circles.[4] As with most programming languages it is important to name your variables. Moreover nodes are able to have labels in order to group them. As we see in the above example in Fig. 2.2 that there are two nodes with the label Person and

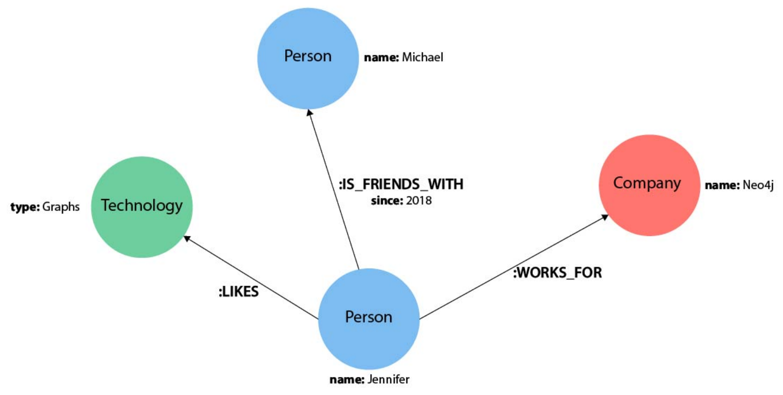


Figure 2.1: *Example of Nodes and Relationships in a Property Graph*

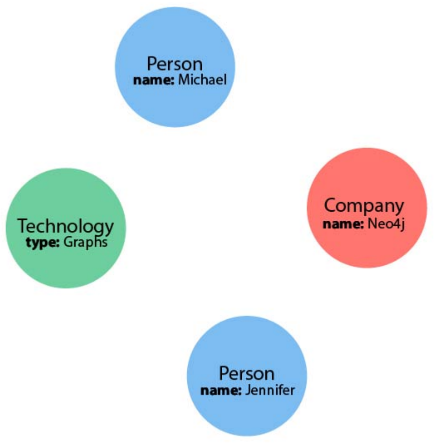


Figure 2.2: *Example of Nodes*

they are represented visually with the color blue to belong to the same group. If we attempt to relate labels to SQL, it could be seen as the table name in some cases.

Examples of specifying nodes based on Fig. 2.2

```
()                    //anonymous node (no label or variable) can refer to any node in the database
(p:Person)            //using variable p and label Person
(:Technology)         //no variable, label Technology
(work:Company)        //using variable work and label Company
```

The second part of graphs are the edges between the nodes, in *Neo4j* these are called relationships.

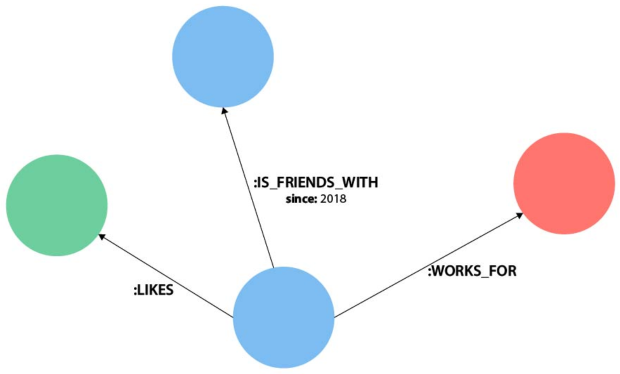


Figure 2.3: *Example of Relationships*

Continuing with readabilit in mind, *Cypher*'s syntax for relationships are arrows as so,

$$--> or <-- .$$

This can be seen in the above Fig. 2.3. Relationships can hold information on their types. These can be placed within the arrows in square brackets.[4]

Some examples to follow:

//data stored with this direction

$$CREATE(p : Person) - [: LIKES]-> (t : Technology)$$

//query relationship backwards will not return results

$$MATCH(p : Person) <-[: LIKES] - (t : Technology)$$

//better to query with undirected relationship unless sure of direction

$$MATCH(p:Person) - [:LIKES] - (t:Technology)$$

The second example would not be favorable if you are attempting to find technologies that a certain person likes it would return no results. Its best practice if you are unsure of the relationship direction to use the third example and have an undirected relationship (which would correspond to an undirected graph).[4]

Similar to labels grouping different nodes together as sen in Fig. 2.2, relationships have different types to group them as well. As with nodes, variable names can be used to refer to a specific relationship. Some examples are as follows. It is important to include the colon in front of the relationship type for the interpreter not to confused them with variable names.[4]

$$[:LIKES]$$

makes sense when we put nodes on either side of the relationship (Jennifer LIKES Graphs).

$$[:IS\_FRIENDS\_WITH]$$

makes sense when we put nodes with it (Jennifer IS_FRIENDS_WITH Michael)

$$[:WORKS\_FOR]$$

makes sense with nodes (Jennifer WORKS_FOR Neo4j)

Lastly, *Cypher* and *Neo4j* allow both nodes and relationships to contain properties. Properties are stored as (key,value) pairs to either nodes, relationships or both where needed.

Using the same graph as before the following are some examples of properties.

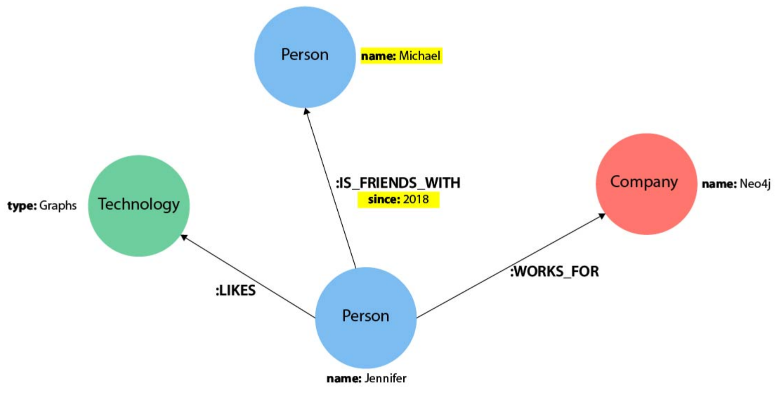


Figure 2.4: *Example of Properties*

Node property:

$$(p : Person\{name :' Jennifer'\})$$

Relationship property:

$$-[rel : IS\_FRIENDS\_WITH\{since : 2018\}]- >$$

We've discussed at this point the main three data models for property graphs.

2.4 SGE

It is important we introduce the SGE accounting format for storing information of submitted jobs on Beocat as previous work in this domain used this software to keep track of data. SGE would keep track of many parameters of a job once they finished running. The many recorded parameters and their meaning can be found on the official Ubuntu manpage.[5]

SGE had other purposes besides accounting. It was also responsible in managing the scheduling of the jobs in the cluster. It would manage its master thread while executing worker threads to run the jobs submitted into the cluster (Beocat in our case). SGE could also provide the administrator with usage information, a list of submitted jobs, varied settings for different scheduling algorithms and access to restart or cancel jobs.[5] SGE was used to manage Beocat's submitted jobs and resource allocations until the end of 2017.

2.5 SLURM

Kansas State University's Beocat Administrator decided to switch over to SLURM from SGE starting January 1st, 2018. SLURM is a linux workload manager similar to SGE but with additional features being kept logged in its accounting manager.[6] SLURM commands at a Beocat head node is used by the user in order to submit jobs, view the status of jobs among other things. An administrator is able to set default values to submitted jobs such as minimum CPUs, minimum required memory and other parameters.

2.6 Graph Mining

It is first important to make the distinction between graph mining and data mining. "Whereas data-mining in structured data focuses on frequent data values, in semi-structured and graph data mining, the structure of the data is just as important as its content"[7]. This is important as a lot of the research this thesis develops is the translation of flat data or tabular data into a property graph format. Another important statement is made by Professor Gudes: "We study the problem of discovering typical patterns of graph data. The discovered patterns can be useful for many applications, including: compact representation of the information, finding strongly connected groups in social networks and in several scientific domains like finding frequent molecular structures"[7]. The salient point for this thesis is finding strong connected groups in social networks as part of this principle is applied in role extraction. Pattern matching is also the main query principle for *Neo4j*'s implementation of property graphs.

2.7 Feature Extraction

Feature extraction is seen as an important first step when presented with data for a machine learning task. Feature extraction is normally preceded by feature selection. As will be seen in the coming chapters the data we are dealing with in this thesis has over 100 different features. Not all features are relevant to a given task and having a process to follow in order to reduce the number is key. What is feature selection then? It is the "problem of selecting some subset of a learning algorithms input variables upon which it should focus attention, while ignoring the rest. In other words, Dimensionality Reduction"[8].

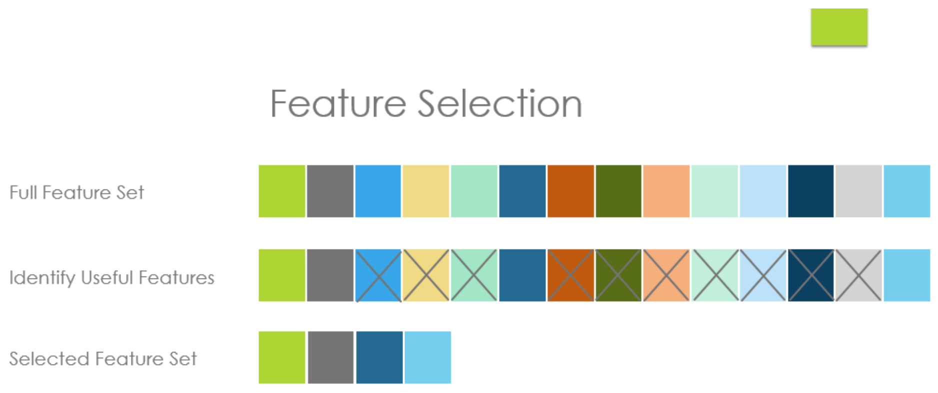


Figure 2.5: *Example of Feature Selection*[8]

As seen in the above Fig. 2.5 a feature selection algorithm was applied to the original set and a subset that maximizes the learning task at hand is selected.

The two main reasons feature selection and feature extraction are done early onto a data set are that they can both improve the machine learning algorithms outcome significantly and when dealing with a lot of variables dimensionality reduction is key.[8]

To select a locally optimal subset of features given a scoring function, enumeration or greedy search is used to maximize the score. According to Mehul Ved's article "The best one can hope for theoretically is the Bayes error rate".[8]

Finally, optimality with respect to a figure of merit should be evaluated in the context of relevance to the prediction objectives. As an example, consider the HPC domain of

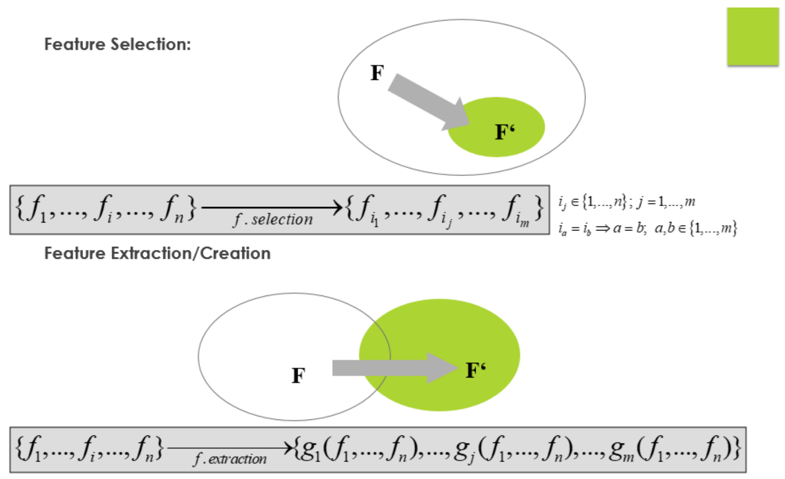

Feature Selection:
F
F'
$\{f_1,\dots,f_i,\dots,f_n\} \xrightarrow{f.selection} \{f_{i_1},\dots,f_{i_j},\dots,f_{i_m}\}$
$i_j \in \{1,\dots,n\};\ j = 1,\dots,m$
$i_a = i_b \Rightarrow a = b;\ a,b \in \{1,\dots,m\}$
Feature Extraction/Creation
F
F'
$\{f_1,\dots,f_i,\dots,f_n\} \xrightarrow{f.extraction} \{g_1(f_1,\dots,f_n),\dots,g_j(f_1,\dots,f_n),\dots,g_m(f_1,\dots,f_n)\}$

Figure 2.6: *Example of Feature Selection and Feature Extraction*[8]

application for this thesis. If the target prediction is memory usage, CPU parameters would have little contribution to the prediction. A given feature selection and extraction algorithm could grant a maximized score in with CPU features but this doesn't mean the feature should be included. The definition of relevance used in this thesis is the degree of error between a given subset of features given by the optimal algorithms versus features that are relevant to the predicted feature.

2.8 Role Extraction

After the optimal set of features is selected, we will study how to extract roles. Role extraction is often seen as a related problem to community detection. Community detection refers to the procedure of identifying groups of interacting vertices (i.e., nodes) in a network depending upon their structural properties (Yang et al., 2013[9]; Kelley et al., 2012[10]). On the other hand, role extraction deals with solving a specific problem faced with certain graphs when applying community detection algorithms.

Meng Wei, a graduate student at Florida State University set the stage for role extraction as follows. "Community structures is not always representative for the structural distribution of nodes of a graph. Bipartite and cyclic graphs are structured but they are not community structures. To generalize community detection, we can resort to the role extraction problem or it is sometimes called block modeling problem."[11] Furthermore, Meng goes to define role extraction as "The goal of role extraction is to reduce a large, potentially incoherent network to a smaller comprehensible structure that can be interpreted more readily. The smaller structured graph is called the reduced graph where nodes are grouped together in roles based on their interactions with nodes in either the same role or different roles. The role extraction problem is a generalization of the disjoint community detection problem where each node in a community mainly interacts with other nodes within the same community.

To classify two nodes as similar, I follow the criterion developed by Koutra et al.: "Two nodes u and v are considered structurally equivalent if they have the same relationships to all other nodes."[12]

Role extraction, as an empirical procedure, is based on the idea that units in a network can be grouped according to the extent to which they are equivalent, according to some meaningful definition of equivalence, such as structural and regular equivalence etc."[11]

Chapter 3

Related Work

In this chapter we will discuss in greater detail the previous work done by Kansas State University in the HPC Analytics domain, and explore graph mining techniques used in the experiment to be discussed in a later chapter.

3.1 Interactive Recommendations by Combining User-Item Preferences with Linked Open Data

When I first started looking at different graph mining procedures and algorithms that could be applied to the data I had initially started with meta walks as many previous papers and work done by other researchers used these methods.

One paper that seemed applicable was ”Interactive Recommendations by Combining User-Item Preferences with Linked Open Data” by Kallumadi and Hsu (2018).[13] It specifically tackles how user based information of linked data can be used for recommendation systems. As stated earlier one of the goals of the HPC project is to create a user-oriented recommendation system to help maximize resource use. The graph mining technique used in Surya Kallumadi’s experiment is meta paths. He describes a meta path to be ”a composite relationship that consists of ordered of edge types specified in the HIN (Heterogeneous Information Network) schema”.[13] However interesting it would have been to use this strategy to find patterns among different users in the beocat data, I had not at this point had a straightforward translation from the log data into a Heterogeneous Information Network. Because of this reason this method was dropped from the experiment, but building a graph seemed to be the best way to go about learning patterns and structure the data in order to improve prediction and lead us closer to our goal of a recommendation system.

Different graph based databases were studied at this point. As will be covered later on, property graphs are the chosen type. RDF and knowledge graphs were also explored. Bare RDFs, lacking the internal structure for both nodes and edges, did not seem sufficient for the role extraction experiments specifically addressed in this work. Our data also did not fit a traditional semantic rule-based solution. RDFs are an efficient and flexible representation for the semantic web domain.[14]

3.2 Machine Learning for Predictive Analytics for Compute Cluster Jobs

When I joined the project there had been a published paper by fellow research assistants tackling the issue of predicting job failures using SGE Data. Their are a few important takeaways from this initial experiment that would help shape decisions in the future using the newer data set. The problem statement as is in this paper "there does not yet exist software that an help to fully automate the allocation of HPC resources or to anticipate resource needs reliably by generalizing over historical data, such as determining the number of processor cores and the amount of memory needed as a function of requests and outcomes of previous job submissions"[1] is still the very same goal that this thesis wishes to aid in by adding in roles as a feature to the existing data. Another feature that the team has been working on is a way to answer "how much would it cost to migrate the job from beocat to AWS or other compute cloud services"[1], this is a secondary goal for this thesis' outcomes. The experiment design for this paper had two main components, regression and classification. The regression tasks were used to estimate the "quantity of resources used (CPU cycles and

RAM)"[1] meanwhile classification tasks would answer "the yes-no question of whether a job will be killed".[1]

Furthermore, this experiment takes an extra step and uses algorithms that construct features for user modeling. This idea of grouping users based on different parameters (given roles such as faculty, graduate, postdoc, staff, undergraduate, etc) is carried on in my thesis and expanded using RolX to find non-explicit roles across different users.

Lastly, this paper states that a long-term goal of this research is to "identify such communities (expertise clusters) by applying network analysis (link mining or graph mining) algorithms to linked data such as property graphs of users and their jobs.[1] This long-term goal is put into practice in the work to be stated in the following chapters of this thesis.

Reader is encouraged to read Machine Learning for Predictive Analytics for Computer Cluster Jobs[1] Evaluation, Conclusion and Future Work for details on performance.

3.3 Improving HPC System Performance by Predicting Job Resources via Supervised Machine Learning

This paper is the second published work in the HPC analytics domain from the research team tackling this task. It is important to first state that the data set used in this paper is from SLURM and not SGE. It also takes the testing of the supervised learning model further by testing it in a SLURM resource manager simulator. Prediction of required memory resources and run time are the main two targets for this paper.[2] The Slurm simulator was developed by the Center for Computational Research, SUNY University at Buffalo. The team had decided to use this implementation of the simulator because "it is implemented from a modification of the actual Slurm code while disabling some unnecessary functionalities which does not affect the functionality of the real Slurm"[2].

The testing and experimentation portions of this paper included, "running each testbed using user requested memory and run time, running each testbed using the actual memory usage and duration, and running each testbed using predicted memory and predicted run time"[2].

The last major takeaway is the additional predicted feature this paper had compared to Machine Learning for Predictive Analytics for Compute Cluster Jobs[1]; predicting time required, a mandatory field a user must input when submitting a job to Beocat. In section 4.3 of this paper the authors state the question "Which is more important to predict? Required memory or required time?"[2] They conclude that both memory prediction and time requested prediction offer equal importance in performance. "Peak performance and utilization was achieved by combining both of them (predicted memory and predicted time) in one model"[2]. For the purpose of this thesis, memory prediction will be the main goal with the added role features.

Reader is encouraged to read Improving HPC System Performance by Predicting Job Resources via Supervised Machine Learning[2] Evaluation, Conclusion and Future Work for details on performance.

3.4 Graph Mining Using Recursive Structural Features

This section will summarize the important algorithms and work related to a KDD 2011 paper that would influence the over arching algorithm used for graph mining the Beocat data set.[15] The authors of the paper state two novel contributions to their work. "A ReFeX (Recursive Feature eXtraction) is scalable and it is effective, capturing regional ("behavior") information in large graphs."[15] Extracting features from a given node that are meaningful is important for many different graph mining tasks, considerably in clustering. As part of the goal in this thesis is to find similarities between (user,job) tuples that could lead to understanding how different users fall into roles. They state the "key step is to extract effective and transferable features from each node that would best capture node characteristics, so that we can distinguish and classify the nodes (or edges)".[15] The algorithm takes in as input a graph G, and delivers as output a node-feature matrix F. The node feature matrix F has two properties. First it is structural meaning it doesn't require information not already found in the graph. Secondly it is effective, the features that are extracted need to help predict other properties of a given node (edge) and also be transferable to other graphs.

Furthermore there are a few of definitions in the proposed algorithm for ReFeX that need to be outlined for this thesis. The goal for ReFeX is to create new recursive features based on what already exists in the graph network. The actual structure of the graph is key in extracting these features, the paper divides the features in to three main attributes, local, egonet and recursive features.[15].

Local and egonet attributes belong to what is called neighborhood features. These are features native to the graph structure itself and include the degree's of both in and outward edges as well as the total degree for a given node. Egonet refer to sub graphs. In our thesis this subgraphs will be comprised of a given user's jobs being linked together. Similar to local attribute, egonet attributes will measure the degree's and count of edges leaving and entering the subgraph.

Recursive features are defined as "any aggregate computed over a feature value among a node's neighbors".[15] There are two main steps with recursive features, generating and pruning. When generating recursive features "ReFeX collects two types of recursive features: means and sums"[15]. Moreover the "features that can be aggregated are not restricted to neighborhood features, or even to structural features".[15] Pruning is necessary as recursive features could be infinite and the algorithm needs a way to define a threshold for a given network. The pruning methods eliminates features that are have a correlated feature pair above a threshold declared by the user.

3.5 RolX: Structural Role Extraction and Mining in Large Graphs

This section will summarize the RolX algorithm first discussed in Henderson, Gallagher, Eliassi-Rad, et al. (2012); it relates back to the algorithm of Section 3.4 as its output is used

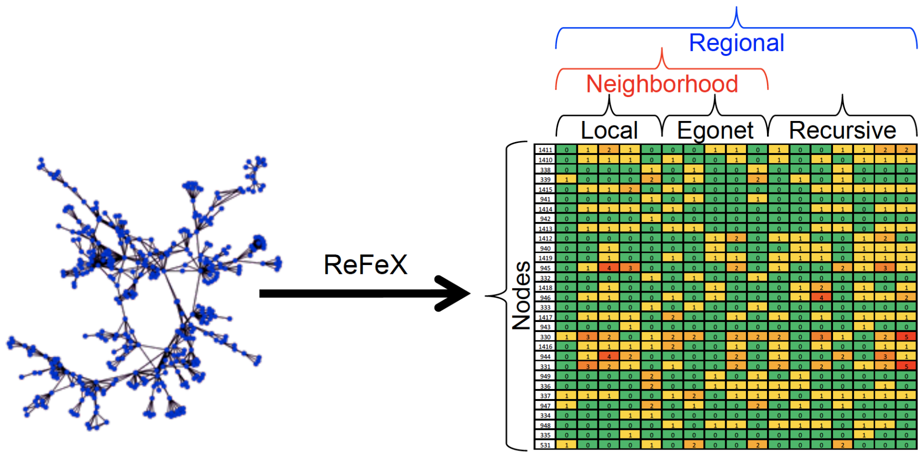


Figure 3.1: *Recursive Feature Extraction*[16]

as input for it.[3] I will also explain the main features and significance that roles bring to the overall research. This is the main contributing paper to this thesis.

The paper starts out by defining what roles are; they capture the structural behavior of a graph. They propose an algorithm, RolX, which is built from the research discussed in Section 3.4. The algorithm is described ”an unsupervised learning approach for automatically extracting structural roles from general network data sets.”[3] They suggest a possible usage for roles is node classification, one of the tasks this thesis wishes to accomplish with adding roles as a feature to the data set. Furthermore the paper describes the main differences between role discovery and network community detection, a related graph mining technique.

The following example shows the major difference between these two techniques.

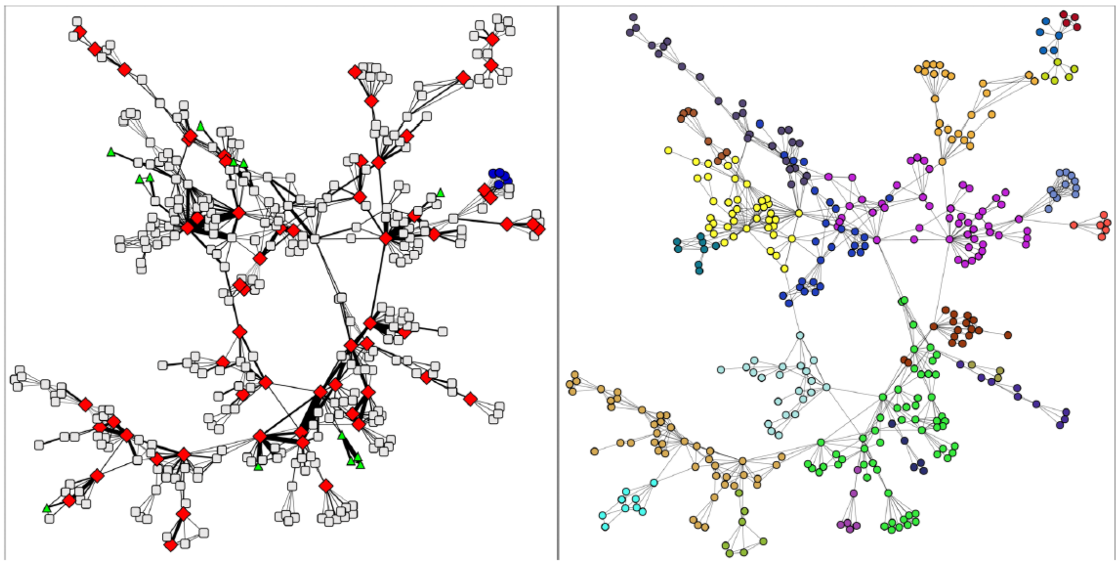


Figure 3.2: *Role Discovery vs. Community Discovery*[3]

Figure 3.2 depicts the difference between role discovery and community discovery for the largest connected component of a weighted co-authorship network[17]. RolX automatically discovers 4 roles vs. the 22 communities that the popular Fast Modularity[18] community discovery algorithm finds.

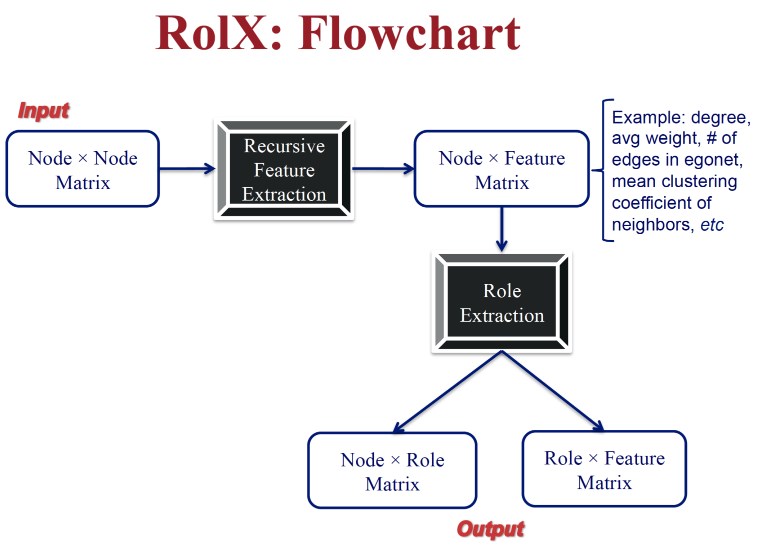

RolX: Flowchart
Input
Node × Node Matrix
Recursive Feature Extraction
Node × Feature Matrix
Example: degree, avg weight, # of edges in egonet, mean clustering coefficient of neighbors, etc
Role Extraction
Node × Role Matrix
Role × Feature Matrix
Output

Figure 3.3: *RolX Flowchart*[16]

A few advantages RolX has over other algorithms are: mixed-memberships over roles, full automation, the use of structural features, generalizability across disjoint networks, and scalability in the number of edges. (RolX run time linear to number of edges).

Chapter 4

Methodology

In this chapter we will outline the process used to collect the data, process the data and designing and executing the experiment.

4.1 Experiment Flowchart

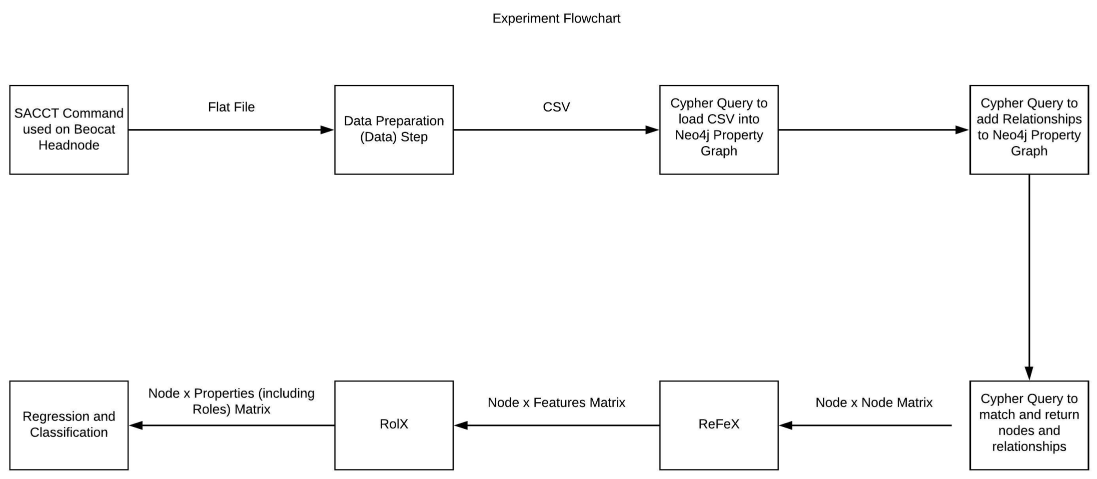
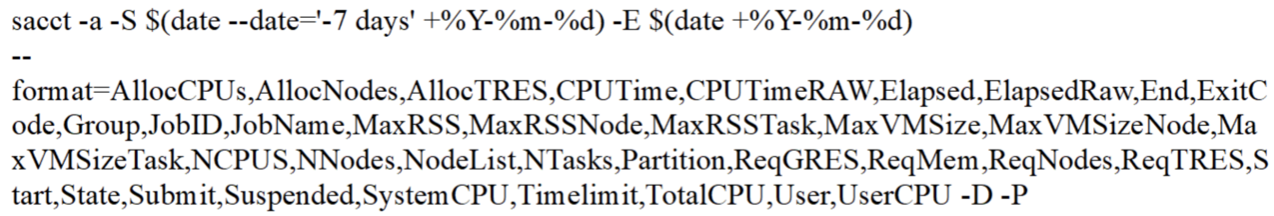

Experiment Flowchart
SACCT Command used on Beocat Headnode
Flat File
Data Preparation (Data) Step
CSV
Cypher Query to load CSV into Neo4j Property Graph
Cypher Query to add Relationships to Neo4j Property Graph
Cypher Query to match and return nodes and relationships
Node x Node Matrix
ReFeX
Node x Features Matrix
RolX
Node x Properties (including Roles) Matrix
Regression and Classification

Figure 4.1: *Experiment Flowchart*

4.2 Data Collection

We will discuss the process for data collection for this thesis in this section. Beocat SLURM data was collected by using the 'sacct' command on a head node of Beocat.

The following command was run to gather the data. It was saved in a slurmData.txt (roughly a 6.8 GiB file). Slurm outputs a — separated file in ISO-8859-1 encoding.



```
sacct -a -S $(date --date='-7 days' +%Y-%m-%d) -E $(date +%Y-%m-%d)
--
format=AllocCPUs,AllocNodes,AllocTRES,CPUTime,CPUTimeRAW,Elapsed,ElapsedRaw,End,ExitC
ode,Group,JobID,JobName,MaxRSS,MaxRSSNode,MaxRSSTask,MaxVMSize,MaxVMSizeNode,Ma
xVMSizeTask,NCPUS,NNodes,NodeList,NTasks,Partition,ReqGRES,ReqMem,ReqNodes,ReqTRES,S
tart,State,Submit,Suspended,SystemCPU,Timelimit,TotalCPU,User,UserCPU -D -P
```

Figure 4.2: *Sacct Command*

The shape of this data set is (9856170, 105) row, column order. This will soon be cleaned and reduced

4.3 Data Preparation

Data preparation for this experiment comes into two different steps. The first data preparation is for transferring a flat text file data to a property graph. This section will correspond with the first of the two data preparations, ad the second step will be in the Experiment Design section. Before I was able to use the data some data cleaning principles were applied.

Firstly we limited the columns to the features that are needed for the experiment. For the purpose of this thesis we will be using memory, run times and roles. Columns are reduced to the set of State, MaxVMSize, ReqMem, TimeLimit, User, Account to be loaded into the graph database. Next a Gaussian distribution mean is found for the data set jobs amounts greater than that mean per user are kept. This dropped the data set to a size of (3367948, 105). Next to keep null values at a minimum I filtered and dropped all rows that don't meet the 4/6 threshold of non null values, meaning only rows with 4 out of the 6 features available were kept. This helped reduce the size of the data set further for the experiment.

Memory needed to be standardized across all jobs. The raw data from SLURM was in string format. It had the substring "mem=" then a number corresponding to the allotted size followed by G, g, T, t, M, or m. I decided to standardize it all to GiB as most of the jobs were requesting memory in this prefix.

I decided with translating each row in the raw data set into a node. The ID for the node was initially be set as a (User, JobID) tuple, which will later on be changed to a unique Node ID.

4.4 Property Graph

With each node representing a unique (User, Job) tuple loaded into Neo4j the next step was to create indexes on the varied properties for faster lookup and queries.

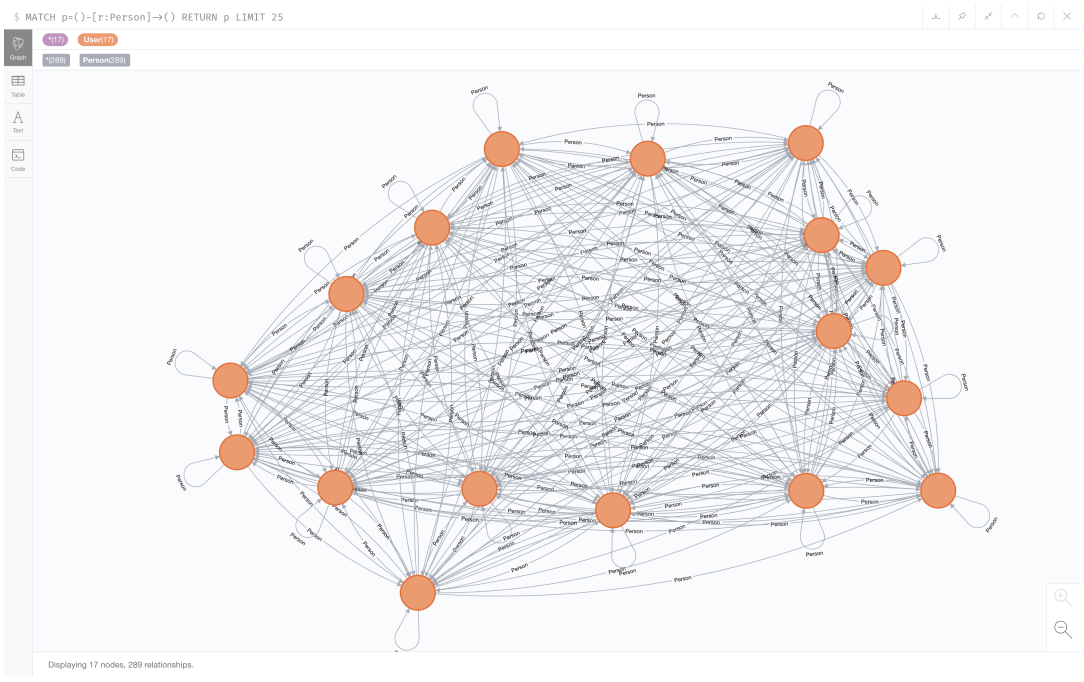
The next challenge was to create relationships between the nodes that would help the feature extraction and role extraction steps to follow. Since we are attempting to learn how different jobs can be classified together into roles I decided to translate a group by statement used in databases into relationships built into the graph. This was done by creating edges between nodes that shared the same User field. (this intuition was brought upon creating a link for grouping all jobs submitted by a User being the same as a group by). Having this relationship already built into the graph made the query's for sampling the data easier.

The following is an example of a person relationship for a given user.

In order to not have a large amount of small linked nodes, I decided on a threshold of a minimum of 4 nodes needed to be connected with edges in order to be kept. Also due to memory constraints on the exponentially increasing number of edges for a User with over 2500 jobs submitted, those nodes are also excluded for the purposes of this experiment.

4.5 Algorithms

Algorithms used in this experiment are also split into two different categories. First are the algorithms used to assign roles to nodes. Second are the algorithms used in regression and


```
$ MATCH p=()-[r:Person]→() RETURN p LIMIT 25
```

Displaying 17 nodes, 289 relationships.

Figure 4.3: *Person Relationship*

classification.

4.5.1 ReFeX and RolX

For both ReFeX and RolX, a PyPi package was used called GraphRole[19]. It's an automatic feature extraction and node role assignment for transfer learning on graphs; based on the ReFeX/RolX algorithms [15,3] of Henderson, et al which are the main two algorithms described in greater detail and used in this thesis mentioned earlier in Sections 3.3 and 3.4.

Graphrole has a method call for each of the algorithms; RecursiveFeatureExtractor and RoleExtractor respectively. Both methods take multi graphs defined by networkx which is a "Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks" [20] as input and output either a dictionary or dataframe. For "reading an entire graph from a Neo4j store into a NetworkX graph" [21] code from a blog post was used authored by Solastisism.

4.5.2 Regression and Classification

The following algorithms were used in the regression tasks. All built into sklearn[22].

Linear Regression, Lasso Model Fit with Akaike information criterion model, Elastic Net model with iterative fitting along a regularization path, Ridge (Linear least squares with l2 regularization) and a decision tree regressor.

The following algorithms were used in the classification tasks also built into sklearn[22].

Logistic Regression (aka logit, MaxEnt) classifier, a decision tree classifier, Gaussian Naive Bayes and a random forest classifier.

These algorithms were chosen to coincide with the algorithms with the same parameters used in Andresen et. al. paper.[1].

4.6 Experiment Design

The total number of nodes used for the experiment is 60352, and the total number of edges is 41702654. This is after the data preparation and property graph building explained in Sections 4.2 and 4.3. The complete dataset accounted for 4.195852758527585e+127 seconds of CPU Time, meanwhile the experiment dataset accounted for 8.65528575285753e+126 seconds of CPU Time. The difference between experiment set and complete dataset is 3.330324183241833e+127. Experiment comprises of 20.62% of the complete dataset's CPU time.

Hardware used for the experiment consists of a Kaos (4th generation) machine, which are one of the systems used as desktop PCs in the KDD Lab. The specs include an Intel - Core i7-6700K 4 cores CPU, 32GB RAM and an nVidia RTX 2080 GPU. The operating system used for this experiment is Ubuntu 18.04.

One of the challenges faced in this experiment was the memory limitations of the experiment's machine. 32GB of ram limited the number of nodes and edges the recursive feature extraction managed to run on as well as the cypher queries. Being able to increment the swap space on the machine made it possible to achieve a successful execution of the experiment.

4.6.1 Data Preparation

As mentioned earlier in Section 4.2, here I will discuss the data preparation that is undergone after roles are assigned to every node. The data needs to further be prepared to be used as input classification and regression tasks. For both tasks all rows that contained null data were dropped. Next depending on if it was a regression or classification task the data was split for the targeted value Y and features X. The test split was set to 0.2.

The following in Table 4.1 outlines some general statistics for the data set used. 54.04% of the dataset consisted of passed jobs, while 45.96% were failed jobs.

Table 4.1: *General Statistics*

Description	Value
Number of unique users	327
Total Set	60352
Total Passed Jobs	32612
Total Failed Jobs	27740

4.6.2 Central Hypothesis

Roles assigned to (User, Job) tuples grant no quantifiable improvement in precision, recall or accuracy in memory or time limit needed for jobs to finish successfully on Beocat.

4.6.3 Experiment Setup and Execution

Running the first part of the experiment (ReFeX and RolX) is run by a python script

This produced three files. The first file includes the nodes with the properties from the graph. The second file consists of the node role assignments and the third file consists of the node role membership by percentage. These three files are then combined to be run into the classification and regression scripts as input.

All scripts can be found on my personal Github.[23]

Table 4.2: *Features*

Feature	Description
Account	Project assigned to job
Timelimit	What the timelimit was/is for the job
User	The user name of the user who ran the job
ReqMem	Minimum required memory for the job
State	Indicates if job failed or not, 0/1
Role	Role a given job belongs to

Baseline experiments used all features except roles, while other experiments used all six features as seen in table 4.3. The features and their descriptions are found in table 4.2.

Table 4.3: *Baseline Experiment Features v.s. Roles Experiment Features*

Baseline Experiment Features	Roles Experiment Features
Account	Account
Timelimit	Timelimit
User	User
ReqMem	ReqMem
State	State
	Roles

The following tables 4.4, 4.5 and 4.6 show the input features and target feature for the regression and classification tasks. Roles is in bold to signify it is only used in Roles experiments.

Table 4.4: *Classification Task*

Input Features	Target Feature
Account	State
Timelimit	
User	
ReqMem	
Role	

Table 4.5: *Regression Task (Memory)*

Input Features	Target Feature
Account	ReqMem
Timelimit	
State	
User	
Role	

Table 4.6: *Regression Task (Timelimit)*

Input Features	Target Feature
Account	Timelimit
ReqMem	
State	
User	
Role	

I will discuss the metrics and results and how they compare those for previous work in the following chapter.

Chapter 5

Results

This chapter describes the evaluation metrics used, presents the experiment results, and gives a comparative analysis with the baseline experiment for the different algorithms and approaches used.

5.1 Evaluation Metrics

We will list out used evaluation metrics.

For the classification tasks, the evaluation metrics used were accuracy and f1 scores for the trained models.

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

$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$

For the regression tasks, the evaluation metric used was the R-squared value of the fitting line for the given models.

$R^2 = 1 - \frac{\sum_{i=1}^{n} e_i^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$

5.2 Results

Roles were assigned by RolX to nodes. Four roles granted the algorithm the best accuracy. The nodes were split as follows into these roles.

Table 5.1: *Role Assignment Count*

Role	Count
role_0	58220
role_1	1774
role_2	313
role_3	0

When interpreting role_3 having 0 nodes assigned to it, one can look at the data for answers. Many users had 10 or less job entries. This caused those nodes to have similar features extracted. When inspecting the node role membership per percentage many of these nodes were split 25% across all nodes. The algorithm defaults to assigning nodes without a winning membership with role_0.

The following are the results for the classification and regression tasks. To demonstrate the credibility of the experimental results, I ran a 10-fold cross-validation.

Table 5.2: *Classification Baseline v.s. Roles*

Model	Accuracy Baseline	F1 Baseline	Accuracy Roles	F1 Roles
LR	0.5416	0.0740	**0.5441**	**0.0827**
CART	0.5608	**0.2963**	**0.5616**	0.2445
GNB	**0.4775**	**0.6278**	0.4691	0.6265
RF	0.5500	**0.3821**	**0.5641**	0.2520

Table 5.3: *Memory Regression Baseline v.s. Roles*

Model	R Squared Baseline	R Squared Roles
LinearRegression	**-0.16981695**	-0.784853
LLIC	**-0.16981695**	-0.784853
ENCV	-0.11498786	**-0.038747**
Ridge	**-0.16979799**	-0.784568
CART	**-0.59263598**	-1.394255

First looking at the classification task results in table 5.2 , we have a clear winning algorithm being Gaussian Naive Bayes with an F1 Score of 62.78 for the baseline and 62.65 for roles. F1 is a better measure for performance compared to accuracy as it considers precision and recall. It is also important to notice the poor performance of Logistic Regression,

Table 5.4: *Timelimit Regression Baseline v.s. Roles*

Model	R Squared Baseline	R Squared Roles
LinearRegression	-0.00636430	**0.289269**
LLIC	-0.05375437	**0.000812**
ENCV	-0.05375437	**0.288984**
Ridge	-0.00636776	**0.289212**
CART	-3.42268847	**0.164293**

signifying this model was the worst fit in classifying with an F1 score of 7.40 for the baseline and 8.27 for roles. Over all between the baseline scores and roles scores we see minimal significant change.

The regression task results are split in two, memory table 5.3 and timelimit table 5.4. Lets explore memory results first. Negative r squared values were given for both baseline and roles experiments. This means that the models don't fit the data. Given these results there is no significance and no clear winner between the models is chosen. Looking at the timelimit baseline vs roles experiment we can see a slight improvement in r squared in adding roles as a feature as they are non-negative numbers. This being the case, no definite conclusion can be given as the baseline experiments models don't fit the data.

5.3 Comparative Analysis

The comparison analysis will take into consideration the following baselines computed from using the experiment code from Andresen, et. al paper[1] with the SLURM dataset. Our experiment only took into account per-user features as the graph database built had relationship mimicking a group by users.

Our comparison will start off with the State classification results. Among the four models the mean accuracy is 53.25%

Next the comparison between regression tasks with roles (this experiment) and the regression task done with the experiment code from Andresen, et. al paper[1] using SLURM Data. The r squared results from my experiment are negative meaning which means the models are a bad fit. Negative results are possible since regression could do worse than the sample mean in terms of tracking the dependent variable. The preceding step to Role Extraction is the RolX algorithm is Recursive Feature Extraction. The features extracted for each node in the graph are as follows. Total degree, in degree, internal edges, out degree and external degree. The data for the features for each node can be found in the raw data listed on my personal github[23] Relationships for this experiment were limited to internal edges (within users that share a username). This meant that the external edges for all nodes was 0. Having a limited way to see a similar structure across users by adding relationships for other user parameters (department, university, etc.) has yielded this experiment as inconclusive.

Figures 5.1 and 5.2 compare F1 scores across inducers for both experiments.

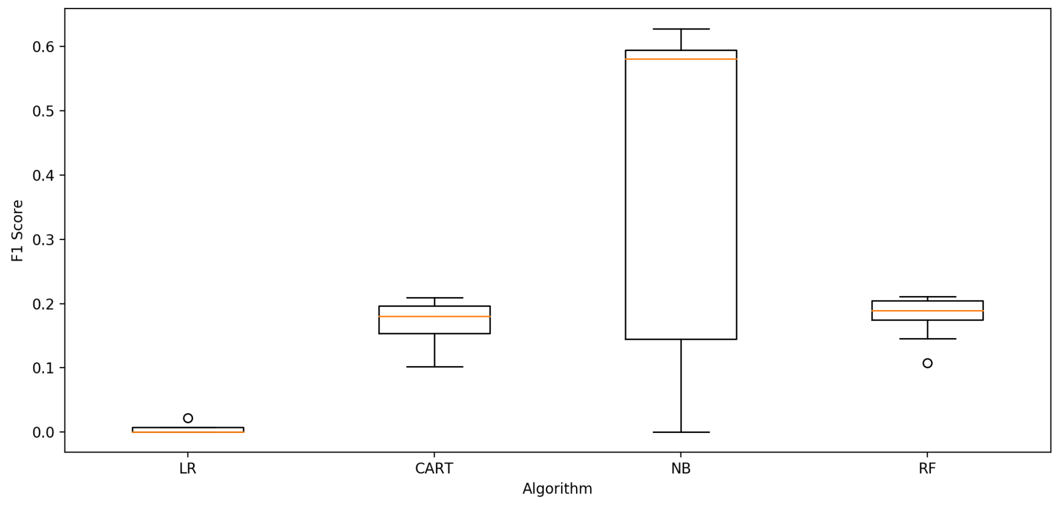


Figure 5.1: *Algorithm Comparison Baseline*

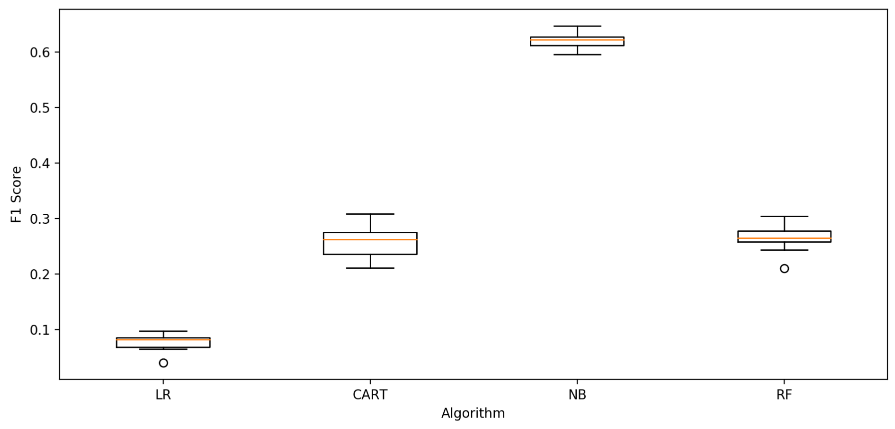


Figure 5.2: *Algorithm Comparison Roles*

Statistical paired t tests were executed for the winning classification algorithm (highest F1 score), Gaussian Naive Bayes, in order to test our null hypothesis.

Hypothesized difference is 0, meaning we expect no change in improvement compared to the baseline.

Table 5.5: *Cross Validation Segment Precision NB*

	1	2	3	4	5	6	7	8	9	10	Mean (μ)	Variance (σ^2)
Baseline Precision	0.4767	0.4857	0.4658	0.4615	0.4387	0.4757	0.4466	0.4929	**0.4806**	**0.4428**	0.4667	0.0003
Roles Precision	**0.7187**	**0.5384**	**0.6000**	**0.5333**	**0.5945**	**0.6521**	**0.4761**	**0.5833**	0.4444	0.3600	0.5501	0.0108

Precision test. The significance level is set to 0.05. I get a t-statistic of -2.672 and a P(T<=t) two-tail result of 0.026. I reject the null hypothesis because p <0.05.

Table 5.6: *Cross Validation Segment Recall NB*

	1	2	3	4	5	6	7	8	9	10	Mean (μ)	Variance (σ^2)
Baseline Recall	**0.9462**	0.8829	**0.9353**	0.9128	0.9608	**0.9611**	**0.9700**	0.9548	0.8706	**0.0383**	0.8433	0.0811
Roles Recall	0.9333	**0.9359**	0.9305	**0.9195**	**0.9629**	0.9580	0.8544	**0.9750**	**0.8918**	0.0330	0.8394	0.0815

Recall test. The significance level is set to 0.05. I get a t-statistic of 0.278 and a P(T<=t) two-tail result of 0.787. I cannot reject the null hypothesis because p >0.05.

Table 5.7: *Cross Validation Segment F1 NB*

	1	2	3	4	5	6	7	8	9	10	Mean (μ)	Variance (σ^2)
Baseline F1	**0.6605**	**0.6343**	**0.6415**	0.5836	0.6053	**0.6443**	**0.6346**	0.0418	**0.6104**	0.6024	0.5658	0.0344
Roles F1	0.6429	0.6269	0.5813	**0.6058**	**0.6245**	0.6192	0.6250	**0.6029**	0.6099	**0.6076**	0.6146	0.0002

F1 test. The significance level is set to 0.05. I get a t-statistic of -0.849 and a P(T<=t) two-tail result of 0.418. I cannot reject the null hypothesis because p >0.05.

Table 5.8: *Cross Validation Segment Accuracy NB*

	1	2	3	4	5	6	7	8	9	10	Mean (μ)	Variance (σ^2)
Baseline Accuracy	0.4666	0.4583	0.4650	0.4950	0.4700	0.4950	0.4850	**0.4700**	0.4533	0.4716	0.4730	0.0002
Roles Accuracy	**0.5000**	**0.5416**	**0.5483**	**0.5516**	**0.5666**	**0.5733**	**0.5400**	0.4683	**0.5333**	**0.5716**	0.5394	0.0010

Accuracy test. The significance level is set to 0.05. I get a t-statistic of -6.688 and a P(T<=t) two-tail result of 0.000. I reject the null hypothesis because p <0.05.

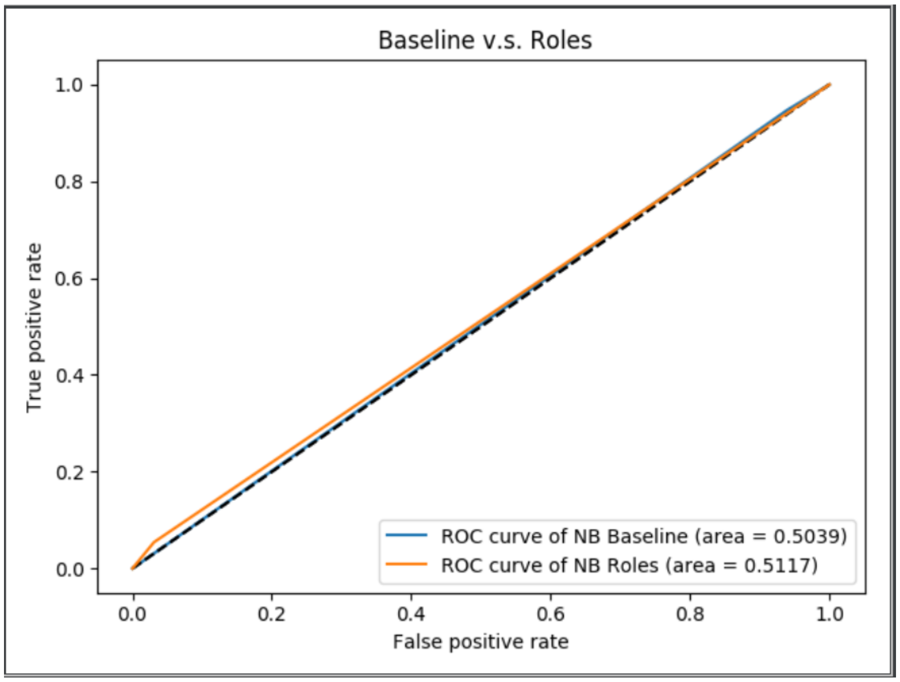


Figure 5.3: *ROC Curve NB Baseline v.s. Roles*

In Figure 5.3 we can see the ROC curve and AUC comparing Baseline v.s. Roles for the Gaussian Naive Bayes Model, so we can see the holistic precision/recall trade off. AUC for the baseline is 0.5039 and AUC for roles is 0.5117. In both cases the AUC is close to 0.5. This AUC score reflects on the classifier not being very useful. This would be a good starting point and there is still a lot of room for improvement. These results show the inducer performing close to random.

Next we will dive into my conclusions and future work.

Chapter 6

Conclusions and Future Work

This chapter comprises conclusions and takeaway lessons that I obtained through the experiments, as well as giving a road map of what is to follow.

6.1 Conclusions

This thesis had as a goal creating a graph feature for the data set from roles. That goal was met and a lot was learned about the current structure of the data, as documented in Chapters 3 and 4. Given the limited mix of roles identified by the algorithm by the limiting external edges this would be a good next step for further improving the graph features. Learning how to use RolX in a new domain led to deriving how easily transferable graph mining can be adapted.

Given the addition of the graph feature, roles, there was no appreciable gain with respect to the baseline. Nevertheless, having implemented a process to create graph features for jobs efficiently gets us one step closer to our end goal of a production level recommendation system for HPC systems.

I performed a series of experiments in order to investigate the impact on accuracy adding roles as a feature could have in our HPC Project when classifying a jobs status (failed or passed) and also predicting time limit and memory usage. As explained in my comparative analysis in Chapter 5, there was no significant change in accuracy, F1 scores, or R-squared compared to the baseline. The results from this experiment are therefore inconclusive and further experimentation with adding more relationships between different users in the graph are needed in order to improve our results.

Comparison to previous work done in this domain by the HPC team is currently not possible as sample sizes and data sets are not comparable. With respect to our null hypothesis, the experiment fails to reject the hypothesis meaning a confidence interval contains a value of no difference; roles assigned to (User, Job) tuples grant no quantifiable improvement in precision, recall or accuracy in memory or time limit needed for jobs to finish successfully on Beocat.

6.2 Future Work

Future work will be consisting in creating more relationship between users be grouping projects, departments, universities among other fields that are currently being unused. Managing a larger sample size in order to run the experiment is also important as it will incorporate more active users with larger job count than ones permitted in this experiment. A colleague of mine at KDD is also working with graphs in order to improve the prediction task of job failures and resources management for Beocat. He is looking into Kernel Graph Convolutional Neural Networks and this would be another possible next step for this project.[24]

Roles as a graph feature granted a new point of view to tabular log data. It allowed for connections to be made between users, and adding more edges will only improve the role membership assignment. For future research what needs to be asked is how can we select user-related features for input as edges in our property graph. Sensemaking is also a set described in the RolX[3] as two different algorithms "one based on node measurements (NodeSense) and another based on neighbor measurements (NeighborSense). These algorithms are to be run in a future experiment. What also needs to undergo sensemaking is the models used in the predictive algorithms. This stems from the limitation of previous work in improving the HPC system using machine learning[2], which yielded uniformly weak regression scores for memory and time limit target variables.

Bibliography

[1] Dan Andresen, William Hsu, Huichen Yang, and Adedolapo Okanlawon. Machine learning for predictive analytics of compute cluster jobs. *CoRR*, abs/1806.01116, 2018. URL http://arxiv.org/abs/1806.01116.

[2] Mohammed Tanash, Brandon Dunn, Daniel Andresen, William Hsu, Huichen Yang, and Adedolapo Okanlawon. Improving hpc system performance by predicting job resources via supervised machine learning. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, PEARC 19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450372275. doi: 10.1145/3332186.3333041. URL https://doi.org/10.1145/3332186.3333041.

[3] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx: Structural role extraction mining in large graphs. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 12, page 12311239, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450314626. doi: 10.1145/2339530.2339723. URL https://doi.org/10.1145/2339530.2339723.

[4] Neo4j. *The Neo4j Cypher Manual v4.0*. https://neo4j.com/docs/cypher-manual/4.0/, 2020.

[5] Inc. Sun Microsystems. *Sge Accounting*, 2008. URL http://manpages.ubuntu.com/manpages/trusty/man5/sge_accounting.5.html.

[6] SchedMD. *Slurm Workload Manager*, 2010. URL https://slurm.schedmd.com/documentation.html.

[7] Ehud Gudes. Graph and web mining - motivation, applications and algorithms, 2018. URL https://www.cs.helsinki.fi/u/langohr/graphmining/slides/chp1.pdf.

[8] Mehul Ved. Feature selection and feature extraction in machine learning: An overview, Jul 2018.

[9] J. Yang, J. McAuley, and J. Leskovec. Community detection in networks with node attributes. In *2013 IEEE 13th International Conference on Data Mining*, pages 1151–1156, Dec 2013. doi: 10.1109/ICDM.2013.167.

[10] Jierui Xie, Stephen Kelley, and Boleslaw K. Szymanski. Overlapping community detection in networks: the state of the art and comparative study. *CoRR*, abs/1110.5813, 2011. URL http://arxiv.org/abs/1110.5813.

[11] Meng Wei. Role extraction, 2020. URL https://www.math.fsu.edu/~mwei/role_extraction.html.

[12] Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Icdm 2014 tutorial node and graph similarity: Theory and applications. URL https://web.eecs.umich.edu/~dkoutra/tut/icdm14.html.

[13] Surya Kallumadi and William Hsu. Interactive recommendations by combining user-item preferences with linked open data. pages 121–125, 07 2018. doi: 10.1145/3213586.3226222.

[14] Pascal Hitzler, Markus Krtzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. Chapman Hall/CRC, 1st edition, 2009. ISBN 142009050X.

[15] Keith Henderson, Brian Gallagher, Lei Li, Leman Akoglu, Tina Eliassi-Rad, Hanghang Tong, and Christos Faloutsos. Its who you know: Graph mining using recursive structural features. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 11, page 663671, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450308137. doi: 10.1145/2020408.2020512. URL https://doi.org/10.1145/2020408.2020512.

[16] Faloutsos Christos Eliassi-Rad, Tina. Discovering roles and anomalies in graphs: Theory and applications part 1 theory, 2012. URL http://eliassi.org/sdm12tut/sdm12tut_roles_part1_final.pdf.

[17] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74:036104, Sep 2006. doi: 10.1103/PhysRevE.74.036104. URL https://link.aps.org/doi/10.1103/PhysRevE.74.036104.

[18] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(6), Dec 2004. ISSN 1550-2376. doi: 10.1103/physreve.70.066111. URL http://dx.doi.org/10.1103/PhysRevE.70.066111.

[19] Daniel Kaslovsky. *Graphrole*. https://github.com/dkaslovsky/GraphRole, 2019.

[20] NetworkX. *NetworkX*. https://networkx.github.io/documentation/stable/, 2020.

[21] Solasistim. Solasistim, May 2018. URL http://www.solasistim.net/posts/neo4j_to_networkx/.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[23] Luis Bobadilla. Python and neo4j used to create a property graph database for ml algorithms - ms thesis, 2020. URL https://github.com/happystep/HPC_Analytics.

[24] Giannis Nikolentzos, Polykarpos Meladianos, Antoine Jean-Pierre Tixier, Konstantinos Skianis, and Michalis Vazirgiannis. Kernel graph convolutional neural networks. *CoRR*, abs/1710.10689, 2017. URL http://arxiv.org/abs/1710.10689.