2018

RETAIL DATA ANALYTICS USING GRAPH DATABASE

Rashmi Priya
University of Kentucky, rashmi.priya@uky.edu
Digital Object Identifier: https://doi.org/10.13023/ETD.2018.221

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Rashmi Priya, Student

Dr. Victor Marek, Major Professor

Dr. Miroslaw Truszczynski, Director of Graduate Studies

RETAIL DATA ANALYTICS USING GRAPH DATABASE

THESIS

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in the
College of Engineering
at the University of Kentucky

By

Rashmi Priya
Lexington, Kentucky
Director : Dr. Victor Marek, Professor of Department of Computer Science
Lexington, Kentucky

2018

ABSTRACT OF THESIS

RETAIL DATA ANALYTICS USING GRAPH DATABASE

Big data is an area focused on storing, processing and visualizing huge amount of data. Today data is growing faster than ever before. We need to find the right tools and applications and build an environment that can help us to obtain valuable insights from the data. Retail is one of the domains that collects huge amount of transaction data everyday. Retailers need to understand their customer's purchasing pattern and behavior in order to take better business decisions.

Market basket analysis is a field in data mining, that is focused on discovering patterns in retail's transaction data. Our goal is to find tools and applications that can be used by retailers to quickly understand their data and take better business decisions. Due to the amount and complexity of data, it is not possible to do such activities manually. We witness that trends change very quickly and retailers want to be quick in adapting the change and taking actions. This needs automation of processes and using algorithms that are efficient and fast. In our work, we mine transaction data by modeling the data as graphs. We use clustering algorithms to discover communities (clusters) in the data and then use the clusters for building a recommendation system that can recommend products to customers based on their buying behavior.

KEYWORDS: Retail Data Analytics, Clustering, Graph Database, Neo4j, Louvain Algorithm, Recommendation System

Author's signature: Rashmi Priya

Date: May 30, 2018

RETAIL DATA ANALYTICS USING GRAPH DATABASE

By

Rashmi Priya

Director of Thesis: Dr. Victor Marek

Director of Graduate Studies: Dr. Miroslaw Truszczynski

Date: May 30, 2018

Dedicated to my father who is not anymore with us but has taught his numerous students, my sister and me to always be thirsty for knowledge and to work hard.

Acknowledgements

My sincerest gratitude to my advisor, Dr. Victor Marek , for his insights and guidance throughout the thesis. He was always happy to meet me and discuss our work. He encouraged and motivated me throughout the process.

I would also like to thank all the professors and especially Dr. Mirek Truszczynski, Director of Graduate Studies, for guiding me throughout my master's study.
Finally, I am thankful to my family and friends who have always supported me, in particular my mother, my sister and my life partner.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today we are surrounded by large networks of data. Some instances of such large networks are genetic and protein interactions networks [38], transportation networks [37], social networks [32] and organizational networks [39]. Such networks contain information that can provide useful insights and patterns. These insights can help organizations take better business decisions. But the size and complexity of such networks makes it difficult to be analyzed manually. For efficient analysis of such large networks, various network analysis tools, such as Stanford Network Analysis Platform (SNAP) [36], have been developed that helps analysts with a huge amount of data to analyze the network efficiently. Network analysis views data as graphs. Graphs consists of nodes and edges. Nodes are the attributes within the network, and edges are the relationships between various attributes. The resulting graph-based structures are often very complex as there can be many kinds of relationships between the nodes.

Another example of such large networks is retail transactional data. Every retail store produces huge amount of transactional data every day. These transactions contain the details of products bought by their customers. Retailers need to analyze this data to find insights and use them to make better decisions for the business. Some examples of useful insights are finding the time of day the customers are more likely to visit the store and assigning more employees for that shift, finding the products that are in bigger demand and filling the inventory accordingly etc. This analysis of retail data has been popularly termed as the market basket analysis [5]. The aim of this study is

to investigate methods and analyze the algorithms that can process and analyze retail data and build a tool that can help the retailers to address such questions.

Many papers have been published that show that transactional information can be used to quite accurately predict customer buying behavior and other aspects of the business such as predicting quantity of a product that needs to be ordered, arrangement of products in the store, finding association rules between products and many more [5][6].

Our work focuses on analyzing retail data to answer some similar questions using graph representations. Graphs have become a popular way of representing complex interconnected systems. The intuitive representation of systems and their ability to be used in the analysis of high dimensional and large amount of data has made graph analytics an exciting new area of study. Graph analytics has become an integral part of analytics today and an integral part of large graph processing systems such as massive distributed graph computing systems including Google Pregel [7] and graph-based machine learning frameworks such as Graphlab [8]. In these systems graph acts as the computational flow model as well as the learning model [6]. Some widely used analysis types includes path analysis, connectivity analysis, centrality analysis and community analysis. We focus on community detection analysis and build a customized product recommendation system [17][15].

This paper is organized as follows: Chapter 2 provides relevant background and graph theory relevant to our work. Chapter 3 describes the methodology used. Chapter 4 describes the application over the Instacart Retail data and Chapter 5 discusses the performance of Neo4j. Chapters 6 presents conclusions and areas for future work.

Chapter 2

Background

In this Chapter, we review relevant literature. Specifically, it is intended to give background and brief introduction to big data and data analytics, the need of non-relational database, various NoSQL databases with the focus on Graph database and relevant graph theory that has been studied in this thesis.

2.1 Big Data and Data Analytics

Big Data is a phrase used to mean a massive volume of both structured and unstructured data. It is a data management challenge that has often been defined by "three V's" i.e., Volume, Variety and Velocity [13][14].

Volume: Volume refers to the amount of data that is generated and collected and it ranges from terabytes to petabytes of data.

Variety: Variety refers to the wide range of data collected from various sources in various formats such as tweets, blogs, videos, music, catalogs, photos etc.

Velocity: Velocity refers to the speed with which the data is generated everyday such as the emails exchanged, social media posts, retail transactions descriptions and the speed with which it is analyzed, whether in real time or near real time.

Companies and researchers are addressing the challenges of big data because it helps organizations to take better decisions and build better products and services. Due to the huge volume of data, especially unstructured data, data storage, analysis and modeling is a clear bottleneck in many applications. The underlying algorithms are not always scalable to handle the challenges of big data. It is also crucial to address the issue of presentation of the results and its interpretation by non-technical domain experts so that the actionable knowledge can be used effectively [10]. These challenges led to the development of data analytics.

The process of extensive use of data to analyze, interpret and communicate decisions and actions is termed as *analytics* [23]. Corporate data has grown consistently and rapidly during the last decade. Due to the availability of the data, analytics has become a crucial part of every business enterprise. The ability to analyze and synthesize information, using information systems, plays an important role in the rising or failure of a company [35].

Traditional analytics rely on a human analyst to generate a hypothesis and test it with a model. The need for consolidating, viewing, and analyzing data according to multiple dimensions, in ways that make sense to analysts at any given point in time gave rise to many analytical technologies. One of the such technology is known as *online analytical processing (OLAP)* [35]. It enables end-users to perform ad hoc analysis of data so that the users can find the insights they need for better decision making. OLAP tools use multidimensional database structures, known as cubes, to store arrays of consolidated information [35].

Analytics has been sub-divided into three categories [11]. First is usually called *descriptive analytics* that describes the past activities, the second is *predictive analytics* that uses the past to predict the future and the third category is *prescriptive analytics* that proposes what needs to be done by performing randomized testing with control groups [11].

Due to big data, businesses wants to collect, store, process, and analyze data in (or almost) real-time and this has been made possible by data analytics. Today analytics is so wide spread that it is not only used by large scale industries but also relatively

small scale businesses. It is being used in almost all domains such as health care, banking, farming, research and others to make better decisions and plan better strategic business moves.

In the next section, we discuss various databases that have been developed to store large amount of data.

2.2 Relational and Non-Relational Databases

Relational databases ruled the information technology industry for more than 40 years. It is a very efficient way of storing and retrieving data that fits into a predefined schema of rows and columns. But in the last couple of decades, the nature of data collection and data processing by computer systems has changed in a number of ways. Due to the increase volume of semi- or unstructured and interconnected data, it has become tedious and inefficient to store data in relational databases as they have strict data model and they deal poorly with relationships [40]. This has led to the need of data models that are less constrained and can still handle huge amount of semi or unstructured and interconnected data. A database that do not follow relational model and is less constraint is termed as *non-relational database* [40]. Both models of databases are widely used today and deciding which database is more suitable for a task at hand is not always trivial as both, relational and non-relational databases, have their advantages and disadvantages. We discuss some of the properties of relational and non-relational databases in the sections that follow.

2.2.1 Relational database

Relational data model was proposed by E. F. Codd in 1970 [26]. It is implemented by a collection of data items organized as a set of formally-described tables and is based on the concepts of relational algebra. The relational model was developed to address a multitude of shortcomings that formerly existed in the field of database management and application development.

Relational database offers powerful and simple solutions for a wide variety of commercial and scientific application problems. Relational database store highly structured data in predetermined *tables*. In every industry, relational systems are being used for applications requiring storing, updating and retrieving of data elements, for operational, transactional, and complex processing as well as decision support systems, including querying and reporting [35]. In relational database, each table contain

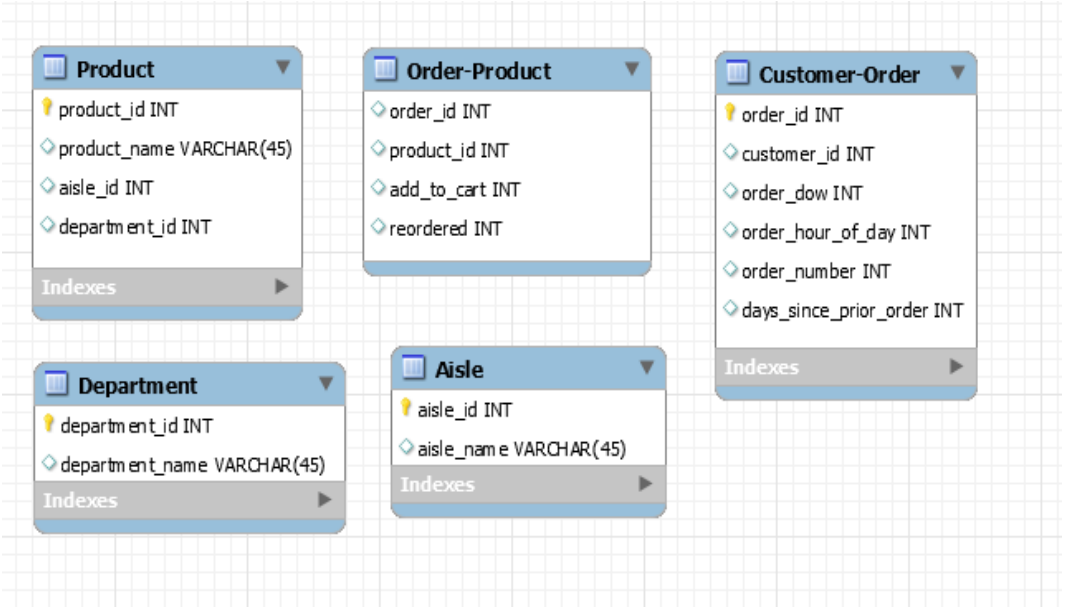

FIGURE 2.1: Example of data implemented by means of Relational Model

rows and columns. Columns represents *attributes* and rows represents *records* in the dataset. Data in a table must possess a primary key that is used to uniquely identify any atomic piece of data within that table and data in different tables are linked through the use of *foreign keys*. Figure 2.1 shows the representation of the dataset used in our work as relational data model [1]. It contains 5 tables named as Product, Order-Product, Customer-Order, Aisle and Department and each table has multiple columns with various details. We will discuss our dataset in a later section.

Relational databases also support transactions that can maintain the consistency of the data even in the event of errors, power failures, etc. It is achieved by satisfying four properties, named Atomicity, Consistency, Isolation and Durability (ACID) [35]. Atomicity property is that all operations in a transaction succeed or every operation is rolled back. Consistency property is that on transaction completion, the database is structurally sound. Isolation property is that the transactions do not contend with one another and transactions *appear* to run sequentially. Durability property requires that

the results of applying a transaction is persistent, even in the presence of any failure such as power or system failure [41].

But relational database has not been very effective in accommodating highly interconnected data as extracting data using queries, that involve large number of joins, is costly [25][40]. With the increase in the usage of the internet leading the need for storing large amounts of semi or unstructured interconnected data, there was a clear desire for a more flexible and effective data store. This led to development of many databases that come in the category of non-relational databases that we discuss in the next section.

2.2.2 Non-Relational Database

Unlike relational database, non-relational database is a category of database that do not follow relational data model to store and retrieve data. It is built on the notion of storing schema free data that can provide faster methods of processing and handling large datasets of unstructured and interconnected data [40]. This category of databases are also referred to as NoSQL databases [47]. The name NoSQL was first used in 1998 by Carlo Strozzi for a shell-based relational database management system [48]. It was a relational database but it did not use the SQL language. The term NoSQL was reintroduced in 2009 as a label for a group of non-relational distributed databases [47]. Today, we have many NoSQL databases available in the market such as Cassandra [45], Mongo [43], Neo4J [21] etc. These databases are based on different models that includes column-based, document-based, key-value-based and graph-based databases that we discuss below. Figure 2.2 shows some NoSQL databases that are widely used by industries and researchers.

Document-based databases: These databases are built on document-based architecture with the capability to store complex data. The term document usually means encoded data in a certain format, for example: XML, JSON, or binary forms like PDF, Microsoft Office documents, etc. MongoDB and CouchDB are two widely used document based databases [43][44]. MongoDB is an open source database that stores collections of documents. It provides high performance data persistence and supports a rich query language, based on JSON, to support read and write operations [43].

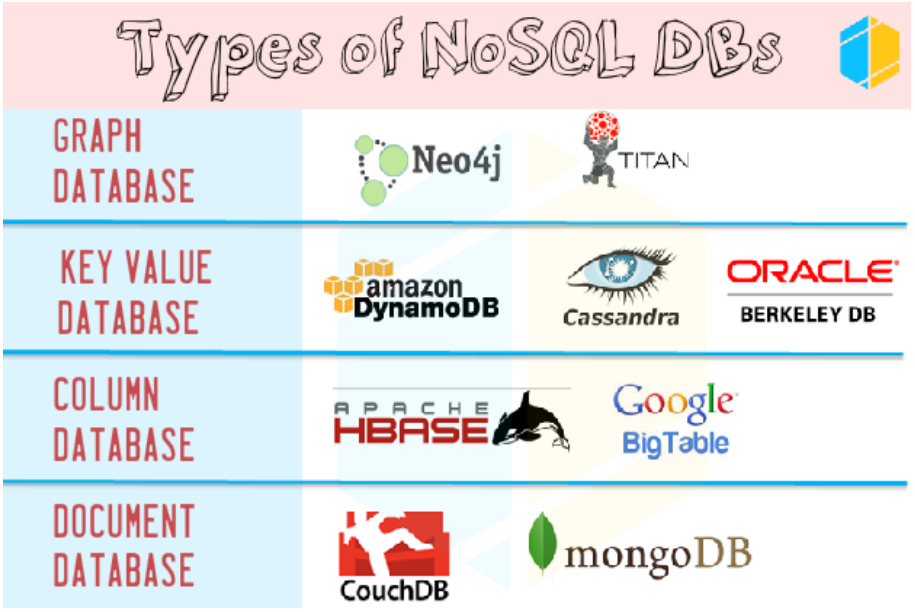

FIGURE 2.2: Examples of NoSQL databases

CouchDB is known to work well with web and mobile applications. It stores data as JSON documents and allows access of documents through the web browser, via HTTP [44]. MongoDB has not yet supported ACID transactions but Couch DB supports them.

Key-Value Stores: These databases store data as key-value pairs, similar to hash table data structure. The values can be of any type and the records are not considered to be related. The data is usually of primitive type (string, integer, etc) or an object defined by certain programming language. This replaces the need for fixed data model and makes the requirement for properly formatted data less strict [12]. As a consequence, it provides scalability and an efficient way of graph partitioning. Oracle NoSQL[50] and Scalaris[49] are specified as key-value databases that allow the application to store schema-less data.

Graph databases: These databases emerged to store relationship-oriented data naturally and efficiently using nodes and edges. Graph databases were proposed on the promises to scale well for large data sets and also simplify the interactions with the applications by reducing the effort needed to map in-memory data structures with databases [12]. One of the reason for the success of graph databases has been the lack of the capability of relational databases to process the highly connected data efficiently

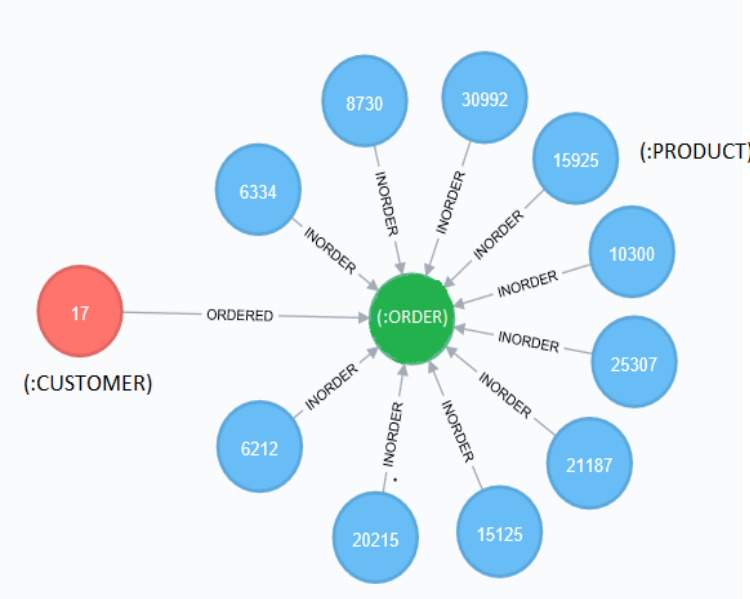

FIGURE 2.3: Graph Data Model in Neo4j

[40]. Various details in the data can also be stored as properties of nodes and edges. It is well-suited for storing complex relationships between attributes. It can offer a cost-saving solution for storing graph-like data, compared to relational model. For example, processing some graph operations to query data from relational databases can be very inefficient, because it may need complex join operations or sub-queries to assist. On the other hand, this type of queries can be easily handled in graph databases. Some graph databases available on the market are HyperGraphDB[46] and Neo4j[21]. Figure 2.3 provides an example representing a part of the data model, of the data used in our project [1], in Neo4j. In this example, we can see three different types of nodes and two relationships. There is a node, representing a customer, that is associated with an order using an ordered relationship and the products in that order are associated by an 'inorder' relationship.

2.3 Relevant concepts of Graph Theory

This section briefly discusses some of the graph concepts and algorithms studied in this thesis.

2.3.1 Centrality

The idea of centrality was introduced by Bavelas in 1948 [55]. Centrality is defined as the relative importance of a node in a graph based on how well they "connect" the graph. For example, in a social network, it is important to know how connected each person is with other people. People who are more connected in a graph are more central to the network and thus they are have more influence or importance in the community represented by that network. There are several different types of centrality measures that have been proposed over the years. Three of the most commonly discussed are degree centrality, closeness centrality and betweenness centrality [16].

Degree centrality is defined as the number of edges incident upon a node (i.e., the degree of the node). If the network is directed, then degree centrality uses two measures. One is in-degree and it is the number of incoming links. The other is out-degree and it is the number of outgoing links. For a graph G(V,E) with n nodes, the degree centrality, $CD(v)$, for node v is defined in [16] as:

$$CD(v) = \frac{degree(v)}{n-1} \tag{2.1}$$

It takes $\Theta(V^2)$ operations to calculate degree centrality for all nodes in a graph in a dense adjacency matrix representation of the graph.

Closeness centrality (or closeness) of a node is the average length of the shortest path between the node and all other nodes in the graph. Thus, intuitively, the more central a node is, the closer it is to all other nodes. Closeness can be regarded as a measure of how long it will take to spread information from a node to all other nodes sequentially. It is defined as the mean geodesic distance (i.e., the shortest path) between a vertex v and all other vertices reachable from it. For a graph G(V,E) with n nodes, the closeness centrality, CC(v), for node v is defined as [16]:

$$CC(v) = \sum_{y} \frac{1}{d(y,v)} \tag{2.2}$$

where d(y,v) is the distance between vertices y and v.

Betweenness centrality is a centrality measure within a graph and can be associated with a vertex or an edge.

Vertex betweenness: Betweenness centrality of a vertex v in a graph G is calculated by summing the fractions of shortest paths between each pair of vertices in G that pass through the vertex v. Vertices that occur on many shortest paths between other vertices have higher betweenness signifying that they are more central in the graph. Betweenness centrality CB(v) for node v is defined as [16]:

$$CB(v) = \sum_{s \neq v \neq t \in V} [\frac{\sigma_{st}(v)}{\sigma_{st}}] \tag{2.3}$$

where σ_{st} is total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v.

Edge betweenness: It signifies total amount of "flow" an edge carries between all pairs of nodes where a single unit of flow between two nodes divides itself evenly among all shortest paths between the nodes. Girvan and Newman efficiently computed edge betweenness in [17] as follows:

foreach *Node n in the network* **do**
    Calculate the breadth-first search starting at node n;
    Determine the number of shortest paths from n to every other node;
    Based on this, determine the amount of flow from n to all other nodes that use each edge;
end
Divide sum of flow of all edges by 2;

Algorithm 1: Edge betweenness

2.3.2 Graph Clustering

Many networks of interest in the sciences, including social networks, computer networks, and metabolic and regulatory networks, are found to divide naturally into communities or modules [16][37]. Detecting groups or communities is an interesting field of study in many branches of science [32] and it is partly attributed to the capability of obtaining and storing large-scale data. The methods for discovering groups in networks have been divided into two main lines of research [29].

One of these areas of research is the applications in parallel computing and integrated circuit design in computer engineering and related areas and is popularly known as graph partitioning [30][31]. The second area is focused on finding the best division of a network that can highlight the underlying structure of large networks which is otherwise difficult to see and is termed as *clustering*, or community structure detection [29][33]. This area is pursued by researchers in many specialties such as Sociology, Physics, Biology, and Applied Mathematics, with applications especially related to social networks [32]. The primary purpose of graph partitioning is to find partitions in the network that have low communication cost and good load balance among computers. This approach has a predefined number of blocks and is based on various criteria and constraints, whereas graph clustering focuses on finding partitions that are strongly connected. The number of obtained, or to be obtained, blocks are mostly unknown with, often, no constraints [33]. As as result of this, clustering methods sometimes yield no good division of the network and this is also considered as an important insight into the data. In this thesis, we focus our discussion on graph clustering.

Vast amount of empirical evidence suggests that large scale graphs such as brain connectivity graphs, protein-to-protein interaction graphs, social networks, transactional graphs, and the web graph, strongly tend to exhibit modularity [5][17]. In other words, they are composed of recursively built clusters, a crucial factor for scaling property. We recall that the process of grouping a set of physical or abstract objects into classes of similar objects is called clustering. The clusters or sub-graphs formed in large

graphs are also termed *modules* (or *communities*). Finding communities helps in understanding various aspects of a graph, for instance localing the set of nodes that exhibit similar behavior and groups that are strongly connected. Community detection has become an integral part in network science as it helps to exploit large networks to understand them better. It is a technique generally used to do initial profiling of the portfolio. After having a good understanding of the portfolio, an objective modelling technique is used to build specific strategy. One of the advantages of clustering is that it generates natural clusters and is not dependent on any driving objective function [3], although objective functions are used to control the quality of communities formed in the clustering process. Clustering is a dynamic field of research in data mining and is categorized as *unsupervised learning* in machine learning [19].

Santo Fortunato suggested that real networks are non-random graphs [15] , as there is a high level of order and organization in these graphs. These large scale graphs tend to form clusters [17]. Each cluster consists of set of nodes that have high density of edges representing more connectivity within each cluster than the number of edges between two clusters [27]. The concept of random graphs was introduced by Paul Erdos and Alfréd Renyi in their 1959 paper, 'On Random Graphs' [18]. The authors defined random graphs as one that has two parameters, one is, n, that is the number of vertices of the graph and the other is p that is the probability of an existence of an edge between any two vertices in the graph. The presence of each edge is statistically independent of all other edges.

The below sections discusses the basic clustering methods, evaluation techniques of clustering and some well-known clustering algorithms.

2.3.3 Basic Clustering Methods

The clustering algorithms are divided into many categories based on their working principles. Here, we discuss the four most widely used categories.

1. *Partitioning methods:* This is most widely used and popular class of clustering algorithms. It divides n data objects into a set of k non-overlapping subsets or

clusters, given k and n such that $k \leq n$. Each data object belongs to exactly one cluster. These algorithms are also known as *iterative relocation algorithms* as these algorithms iteratively relocate data objects between clusters until a (local) optimal partition is found. In basic iterative algorithms, convergence is local and the globally optimal solution can not be guaranteed (a NP-hard problem) [20]. *K*-means is the most widely used and the best-known partitioning-based clustering method [19][20].

2. *Hierarchical methods:* A hierarchical method creates a hierarchical decomposition of a given set of data objects. A hierarchical method can be classified as being either agglomerative or divisive, based on how the hierarchical decomposition is formed. The *agglomerative* approach, also called the bottom-up approach, starts with each data object in a separate group or cluster. It successively merges the closest pair of clusters until all the data objects are merged into one cluster, or a termination condition holds. The key operation of basic agglomerative clustering is the computation of the proximity between two clusters and the definition of cluster proximity differentiates various agglomerative algorithms. For instance *'min single link'* cluster proximity defines it as the shortest distance between two points that are in two different clusters and *'group average proximity'* defines it as average distance between two points. Agglomerative hierarchical clustering algorithms are expensive to compute and require significant storage resources. Louvain community detection algorithm is an example of this technique that we discuss in a later section. The *divisive* approach, also called the top-down approach, starts with all the objects in one, all-inclusive cluster. In each successive iteration, a cluster is split into smaller clusters, until eventually each object is in one cluster, or a termination condition holds. Girvan-Newman edge-betweenness community detection algorithm is an example of this technique that we also discuss in a later section. Hierarchical methods suffer from the fact that once a step (merge or split) is done, it cannot be undone. This prevents global optimization and thus, is not suitable for noisy, high-dimensional data [19][20].

3. *Density-based methods:* The density-based clustering methods discovers clusters

of non-spherical shape unlike partitioning and hierarchical methods that are designed to find spherical-shaped clusters. DBSCAN, OPTICS and DENCLUE are basic techniques of density based methods [19]. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) finds objects that have dense neighborhoods. It connects such objects and their neighborhoods to form dense regions as clusters. OPTICS (Ordering Points to Identify the Clustering Structure) outputs a cluster ordering which is a linear list of all objects under analysis and objects in a denser cluster are listed closer to each other. DENCLUE (DENsity-based CLUstEring) is a clustering method based on a set of density distribution functions [19].

4. *Grid-based methods:* Grid-based methods are space-driven approach rather than data-driven approach. The object space is divided into a finite number of cells that form a grid structure. All the clustering operations are performed on the grid structure. It has fast processing time and is independent of the number of data objects and dependent only on the number of cells in each dimension in the space [19].

2.3.4 Evaluation of Clustering

Clustering must be evaluated before and after the run of the algorithm. Specifically, it is important to evaluate the feasibility of clustering analysis on a data set before clustering is applied and the quality of the clusters after the clusters are generated. In this section, we discuss the major tasks of evaluating clusters as specified by Jiawei Han in his book *Data Mining Concepts and Techniques* [19].

Assessing Clustering Tendency: The application of any clustering algorithm leads to formation of clusters but some of the results can be misleading. Thus, it is important to evaluate the clusters and perform cluster analysis on the data set and it should be considered meaningful only when there is a nonrandom structure in the data [19].

Determining number of clusters: It is important to determine the number of clusters in a data set as it helps to find a balance between compressibility and accuracy in

cluster analysis. But the concept of finding the 'right' number of clusters is ambiguous and it depends on the scenario and the data set. Algorithms like *k*-means sets the number of clusters in a data set as the parameter before applying the technique but some other algorithms does not have such requirement. The number of clusters can be considered as an interesting and important statistic of a data set.

One of the method that helps to find an appropriate number of clusters is the Elbow method that is based on interpretation and validation of consistency within cluster analysis [22]. This technique is based on the observation that increasing the number of clusters can help to reduce the variance within a cluster and thus more similar data objects can be in the same community. For example, if we look at the entire data object as one cluster, not much insight can be gained by it, but creating clusters can help to divide the data objects and bring those communities consisting of 'similar' objects together. But at the same time, if too many clusters are formed, then the effect of reducing the variance within clusters may drop. For example, if we place each data object in its own cluster, then it does not enable any data summarization. Thus, while selecting the number of clusters, it is important to use the turning point in the curve of the sum of within-cluster variances with respect to the number of clusters where the turning point signifies the drop in the marginal gain and use this point as the 'right' number of clusters [22][19]. It is desirable to estimate this number even before a clustering algorithm is used to derive detailed clusters.

Measuring Clustering Quality: Once the clusters are generated, it is crucial to measure the quality of the clusters. A number of measures can be used for this task. Some of these measures are finding how well the clusters fit the data set and such methods are known as *intrinsic* methods [19]. Some of these measure how well the clusters match the ground truth, if such truth is available. Such methods are known as *extrinsic* methods [19]. There are also measures that score clusterings and thus can compare two sets of clustering results on the same data set.

2.3.5 Clustering Algorithms

As a result of the importance of community discovery in developing large graph analytics, various algorithms have been proposed [6]. Modularity maximization is one

of the most popular technique for community detection in graphs. We discuss modularity and some of the algorithms that uses modularity maximization technique in the below sections.

2.3.6 Modularity:

Modularity is a measure that quantifies the quality of the clusters formed in a network by a clustering algorithm [28]. It is based on the idea that true community structure in a network corresponds to a certain arrangement of edges [28]. Modularity is computed as the fraction of the edges that fall within the given groups minus the expected number in an equivalent network with edges placed at random [27]. Modularity is defined as [3]:

$$Modularity(Q) = \frac{1}{2m} \cdot \sum_{i,j} [A_{i,j} - \frac{k_i \cdot k_j}{2m}] \delta(c_i, c_j) \tag{2.4}$$

where:

$A_{i,j}$ represents the weight of the edge between i and j,

$k_i = \sum_j A_{i,j}$ is the sum of the weights of the edges attached to vertex i,

m is the sum of the weight of all the edges in the given network and is computed as $Q = \frac{1}{2} \sum_{i,j} A_{i,j}$,

c_i is the community to which vertex i is assigned and,

$\delta(c_i, c_j)$ is the Kronecker delta function which is 1 if nodes i and j are assigned to the same community, else it is 0.

$\frac{k_i k_j}{2m}$ signifies the expected number of edges between vertices i and j if edges are placed at random.

Thus, the modularity, Q, is given by the sum of $A_{i,j} - \frac{k_i k_j}{2m}$ over all pairs of vertices i, j that fall in the same group.

Modularity is not only used as an objective function to optimize clustering but also to compare clusters obtained by various methods. Networks with high modularity have dense connections among the nodes within the same module and sparse connections between the nodes in different modules. The value of the modularity lies between -1 and 1, [3]. Value other than 0 indicates deviations from randomness and it has been shown in [27][28] that in real networks modularity value greater than 0.3

may indicate significant community structure. Some clustering algorithms maximize modularity while creating clusters. Positive change in modularity indicates that the number of edges within clusters, after the addition of the node to a community, exceeds the number expected on the basis of chance [3][27]. Maximizing modularity is known to be a NP-hard problem [58].

The traditional methods of detecting clustering, first calculates a parameter called *weight*. It can be calculated in different ways, for instance the number of paths in the graph that pass through both the vertices. Weight conceptually represents how strongly two vertices are connected. Then the algorithm takes all the vertices in the network with no edges between them and adds edges to the graph in the decreasing order of the weights. As the edges are added, the resulting graph produces a subgraph of connected components which are emitted as the communities.

The traditional methods of detecting clusters have not been very successful as the algorithm tends to isolate the single peripheral vertices in the network that are connected to the graph by a single edge. This is mainly due to the design of the algorithm.

We discuss here two of the well known community detection algorithms known are Girvan-Newman [34] and Louvain [3] community detection algorithm.

2.3.7 Girvan-Newman Algorithm

Girvan-Newman is often considered the first algorithm of the modern age of community detection in graphs [34]. It is categorized as a hierarchical divisive algorithm. Here the links are iteratively removed based on the value of their betweenness, which expresses the number of shortest paths between pairs of nodes that pass through the link. The idea is that it is likely that edges connecting separate modules will have high edge betweenness as all the shortest paths from one module to another will traverse through them.

The algorithm begins by calculating betweenness for all edges in the network and

removes the edge with the highest betweenness. The algorithm then recalculates betweennesses for all edges affected by the removal and repeats the same process until no edges remain in the network [17]. However, in its most popular implementation, the procedure of link removal ends when the modularity of the resulting partition reaches a maximum [34]. The algorithm has a complexity $O(N^3)$ on sparse graph [17][34]. Table 2.1 shows the pseudocode for Girvan-Newman algorithm.

TABLE 2.1: Girvan-Newman Algorithm [17]

```
while Edges exist in the network do
    foreach Edge in the network do
        Calculate the betweenness of the edge;
    end
    Find the edge with the maximum edge betweenness;
    Remove that edge;
end
```

2.3.8 Louvain Algorithm

Blondel et al. proposed a greedy algorithm to detect communities in large graphs [3]. This algorithm is known as *Louvain method of community detection*. This algorithm is categorized as hierarchical agglomerative algorithm. The algorithm uses heuristic method based on modularity optimization to find communities. The algorithm works in two phases. The first phase is modularity improvement phase. Initially, each node in the network is assigned a separate community, so the number of communities is equal to number of nodes. Then, for each node i in the network, the change in modularity is calculated considering the node i will be moved from its current community to its neighbor's community. Then the node i is moved to one of its neighbour which results in maximum change in positive modularity. The change in modularity is calculated as follows:

$$\Delta Modularity = [\frac{\sum_{in} + K_{i,in}}{2m} - (\frac{\sum_{tot} + K_i}{2m})^2)] - [\frac{\sum_{in}}{2m} - (\frac{\sum_{tot}}{2m})^2 - \frac{K_i}{2m}] \quad (2.5)$$

where:

$\sum_{in}$ in is the sum of the weights of the links inside C,

$\sum_{tot}$ is the sum of the weights of the links incident to nodes in C,

k_i is the sum of the weights of the links incident to node i,
$k_{i,in}$ is the sum of the weights of the links from i to nodes in C,
and m is the sum of the weights of all the links in the network.

If none of the possible moves results in increase in modularity, node i stays in its current community. This process is recursively repeated for all nodes until the process reaches a local maximum and no further improvement can be achieved. Maximizing modularity is an NP-complete problem so the algorithm chooses the local maximum [3].

In the second phase of the algorithm, a new network is built in which the communities, that were found during the first phase, become the nodes. The weight between two nodes (that is two communities) is the sum of the weight of the links between nodes in the corresponding two communities whereas links among nodes in the same community lead to edges that are self-loop, that is their starting and ending points are same. This completes the second phase and this results in a smaller network on which the first phase of the algorithm can be reapplied. These two phases are repeated until phase one is no longer able to obtain any more increase in modularity signifying that the nodes should not be combined with more communities. It is important to note that the modularity is always computed with respect to the original network as the second phases still maintains the number of links in the original network [34]. Louvain community detection is an unsupervised method for detecting communities and its complexity is linear on sparse data [3][34] with most of the computational effort spent on the optimization at the first level. Table 2.2 shows the pseudocode for the Louvain-method.

2.3.9 Limitations of a Modularity Maximization Technique

The algorithms based on modularity optimization have been widely used in practical applications. But modularity optimization technique also suffers from certain limitations. One of the limitation is termed *resolution limit* [57][58]. It has been shown that modularity optimization may fail to identify modules smaller than a certain size, or weight, [59]. This limit is considered to be a consequence of assuming that connections

TABLE 2.2: Louvain community detection Algorithm

```
G be the initial network;
while Put each node of G in its own community do
    while some nodes are moved do
        foreach Node in the network do
            place n in its neighboring community including its own which maximizes
            the modularity gain;
        end
    end
    if the new modularity is higher than the previous modularity then
        G = the network between communities of G;
    else
        Terminate;
    end
end
```

between the communities follow the random graph model. In equation 2.5, we saw that the change in modularity does not depend upon the modules' internal structure. There is a gain in modularity whenever the observed number of edges between two communities exceeds the number expected for a random graph with the same degree sequence. This behavior has been shown to be more prominent in large unweighted graphs because modularity tends to expect the weight be less than 1 while the minimum weight between modules is 1. Weighted networks might be free of resolution limit [57]. Researchers have suggested that analyzing the communities generated using modularity optimization techniques is quite necessary.

Chapter 3

Methodology

The methodology used in our work is based on the CRoss-Industry Standard Process for Data Mining (CRISP-DM) [9]. It is an abstract model proposed by IBM in 1996. The model is independent of industry, tool and application and describes six phases for conducting a data mining project. The six phases are: understanding business, data understanding, data preparation, modeling, evaluation and deployment [9]. The description below follows the outline of [9].

Business understanding

This is the initial phase that focuses on understanding the needs and objectives of a business for which data is mined. Every business is focused on solving a specific problem and thus has a specific set of requirements and needs. While developing an analytical solution for a business, it is extremely important to understand the problem that is being addressed and then build a solution that aligns with that business needs.

Data understanding

Understanding the data starts with the initial data collection and analysis. It is important to get familiarized with the data, identify data quality problems, discover first insights into the data and detect interesting subsets of data that can form hypotheses for hidden information.

Data preparation

The data preparation phase covers all activities needed to construct the final dataset that will be used in the model. Data preparation might have to be performed multiple times based on the facts and findings in the data. Data preparation includes attribute selection as well as transformation and cleaning of data for modeling tools.

Modeling

In this phase, various modeling techniques are selected and applied, and their parameters are calibrated to optimal values. Some well-known modeling techniques are: classification analysis, association rule learning, outlier detection and clustering analysis [51]. Typically, there are several techniques for the same data mining problem type. Some techniques have specific requirements on the form of data. Therefore, stepping back to the data preparation phase is often needed.

Evaluation

At this stage of the project, a model is built that appears to have high quality from the data analysis perspective. Before proceeding to final deployment of the model, it is important to evaluate the model thoroughly, and review the steps executed to construct the model, to be certain that it properly achieves the business objectives. A key goal is to determine if there are some important business issues that has not been sufficiently considered. At the end of this phase, a decision on the use of the data mining results should be reached.

Deployment

Creation of the model is generally not the end of the project. Even if the purpose of the model is to increase knowledge of the data, the knowledge gained in the process will need to be organized and presented in a way that is useful to the customer and thus the model must be successfully deployed.

The aim of our study is to analyze retail data using a graph database and create a recommendation system that can provide personalized coupons to customers. We selected Neo4j [2] for storing and analyzing our data as graphs. It is an open source and widely used graph database that promises to scale well for large datasets. It is ACID-compliant transactional database with native graph storage and processing with simple and powerful data model [21]. We implemented a modularity maximization algorithm, Louvain community detection algorithm [3], for creating clusters in our data. It is a fast algorithm and has been reported suitable for real networks [52][53][21]. We implemented the algorithm in Java. Neo4j provides APIs in Java that helps us to easily connect to Neo4j and execute queries.

In the next section, we discuss the set up, implementation and our understanding before applying on our system.

3.1 Data Preparation in Neo4j

Neo4j can run on any desktop or laptop computer and on all popular operating systems such as Windows, Linux and Mac OS [21]. We used a Windows laptop for downloading and running Neo4j. Before installing Neo4j, one needs Java Virtual Machine (JVM) and Java Development Kit (JDK) installed on the computer. Once we had these, we downloaded the Neo4j from the official website [2], and using our administrative rights we installed it as a Windows service by following the steps provided in Neo4j manual [21]. We use *neo4j start* and *neo4j stop* commands to start and stop the service respectively.

Once we start the service, we can run queries by opening a web browser and going to http://localhost:7474/. The "Data browser" tab allows us to create, query and delete data in the database. We used *Cypher* query language [21] for querying data. Our choice of cypher is motivated by the expressivity and compactness of the graph database query language. The commands are similar to SQL language and with our habit of representing graphs as diagrams, this makes it ideal for programmatically describing graphs. Cypher allows us to find data that matches a specific pattern as

discussed below. Figure 3.1 shows a sample graph created on Neo4j. The graph in

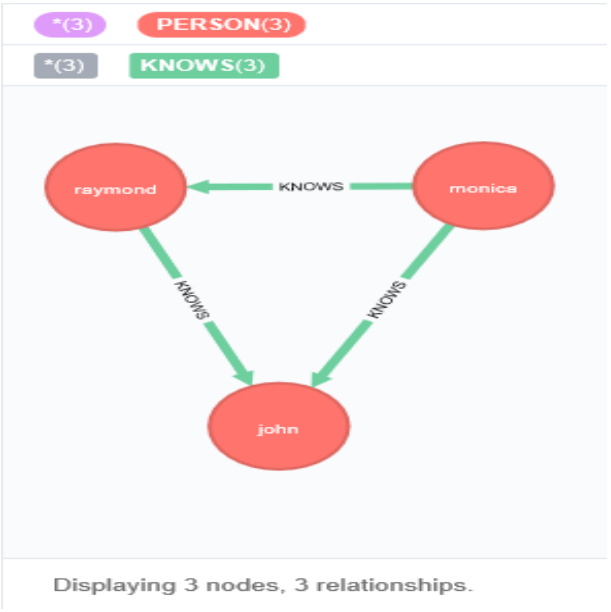

FIGURE 3.1: Example of data represented in Neo4j

figure is written in its equivalent ASCII art representation in Cypher as:

(*john*)<-[:*KNOWS*]-(*monica*)-[:*KNOWS*]->(*raymond*)-[:*KNOWS*]->(*john*)

This pattern describes a path that connects a node, named *monica*, to two nodes, named *john* and *raymond*. It also connects *raymond* with *john*. There are three nodes in this sample graph and *john*, *monica* and *raymond* are the identifiers of their respective nodes. The nodes are connected by an edge or a relationship named *KNOWS*. Identifiers help us to refer to the same node more than once when describing a pattern. For example, we have accessed the node with identifier *john* twice. This example shows that cypher query language can be laid out in two dimensions and the representation is similar to a graph drawn on a whiteboard.

Nodes and relationships can have values and labels that can help to locate the relevant elements in the dataset [40]. For example: in the above example, we can create three nodes with label *Person* that signifies the type of node. The label can have various properties, such as *name* that can store the name of the person.

Cypher is composed of clauses that are similar to that in SQL [40]. The simplest

queries consist of a MATCH clause followed by a RETURN clause. For example, to get the names of the two people known by *Monica* will be similar to a query below:

MATCH (*a*:*Person* name:*Monica*)-[:*KNOWS*]->(*b*:*Perso*n), (*a*)-[:*KNOWS*]->(*c*:*Person*) RETURN *b*, *c*

The MATCH clause is at the heart of most Cypher queries and is similar to the SELECT clause in SQL. The nodes are drawn with parentheses, and relationships using pairs of dashes with greater-than or less-than signs ($--$ > and < $--$). The < and > signs indicate relationship direction. We put the relationship name between the dashes in a square bracket and prefixed with a colon. Node labels are similarly prefixed with a colon. RETURN clause specifies which nodes, relationships, and properties in the matched data should be returned to the client. Some of the most common commands used, while using Cypher, are where, merge, create, create unique and many more.

Neo4j also provides many drivers to connect with other applications [21]. The implementations that are available range from the Neo4j's Java API, the REST interface and a domain-specific, path-based, query language named Gremlin [24]. We used Neo4j's Java Native API to load data and Neo4j's Java Cypher API to query data while building our application. That interface provides easy to use functions, such as createNode(label.labelname), createRelationshipTo(node,relationship name) and setProperty(object, value), for creating nodes, relationships and setting properties respectively [21].

3.2 Algorithm for Clustering

We implemented Louvain community detection algorithm [3], as discussed in Section 2.3.5, in Java. Initially we assigned each node of the graph to a separate community. As Louvain community detection algorithm works in two phases, we initialize two variables, *moves* and *totalmoves* to signify the end of each phase. In the first phase, we calculate the modularity of the graph using Equation 2.4. We then traverse all the nodes of the graph. At each node, we traverse all its neighboring communities. The goal is to find the best community that the node can be moved to. So, we calculate

the change in modularity using Equation 2.5, for each of its neighboring communities. We choose the community which results in the maximum change in the modularity. If the node is not in the community already, the node is moved to the community and we also increase *moves* by 1. If a node is moved, the edges between the nodes in the same community are changed to a self loop as the nodes within a community are considered a single node and we then move to the next node.

When we finish traversing all the nodes, we increase *totalmoves* by moves and calculate the modularity of the new graph. If there has been a change in modularity and there were some moves (i.e., $moves > 0$), we continue traversing the graph as in above paragraph. But if there is no change in modularity or there was no movement of nodes to any community, we transfer to the second phase.

In the second phase, we look at the *totalmoves*. If the *totalmoves* is greater than zero, then the nodes in the same community are changed to one node without changing the overall links. The links within a community are changed to self-loops and the links between the communities are aggregated as the weight of the edge between them. Then we again start with first phase. But if the *totalmoves* is zero, then there has been no improvement in modularity and we consider that this is the maximum modularity that the graph can attain and return the communities.

3.3 Evaluating Clustering Algorithm

We evaluated our clustering algorithm on a graph shown in Figure 3.2. That graph is known as connected *caveman graph* and was first suggested by Duncan J. Watts [54]. It is formed by modifying a set of isolated k-cliques by removing one edge from each clique and using it to connect to a neighboring clique. This graph does not resemble a real network but we chose it as it will help us in evaluating our implementation of the algorithm. We can see that such graph contains communities and we can use them to verify our algorithm. The example we considered was particularly simple and initially we used an unweighted caveman graph for our analysis. We present the results of the community detection in Table 3.1.

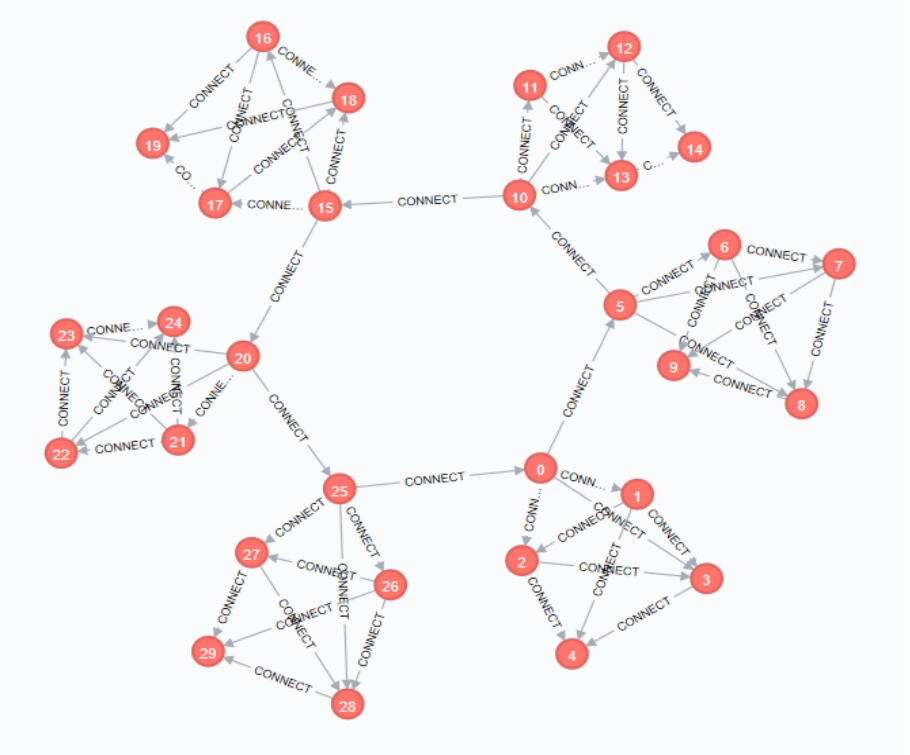

FIGURE 3.2: Connected Caveman Graph of 6 cliques

TABLE 3.1: Communities detected in unweighted Caveman Graphs of different sizes

Cliques	Nodes	Edges	Communities	Modularity
6	30	60	6	0.733
18	90	180	18	0.844
24	120	240	15	0.865

We can see that our implementation resulted in detecting correct number of communities in connected caveman graphs of sizes 30 and 90 nodes. Each of the graphs had 6 and 18 cliques and our algorithm generated 6 and 18 communities respectively. But when we further increased the size of the graph to 120 nodes with 24 cliques, the algorithm generated 15 communities.

To analyze the result formed in the third case above, we looked at the number of communities and modularity formed at each phase of the algorithm. We found that the algorithm started by calculating an initial modularity of -0.0085 with 120 communities. At the end of phase 1, we found the algorithm resulted in 24 communities with modularity of 0.858. But the algorithm further reduced the communities to 15 communities with modularity of 0.865. We can see that the algorithm found the correct number of communities in one of its levels but it further reduced the number of

communities to achieve a higher modularity. This is known as the resolution limit in algorithms using modularity maximization technique [57]. It shows that modularity maximization technique has trouble detecting smaller communities in larger networks but the intuitive community structure is available as an intermediate step in the process.

We also evaluated our algorithm on a *weighted* caveman graphs. We kept higher weighted edges within cliques and lower weighted edges between cliques. Table 3.2 shows the results of the community detection.

Table 3.2: Communities detected in weighted Caveman Graphs of different sizes

Cliques	Nodes	Edges	Communities	Modularity
24	120	240	24	0.950
36	180	360	36	0.964

We found that the algorithm detected the expected communities in the weighted caveman graphs and did not combine the smaller communities. Our finding aligns with the conclusions given by Benjamin H. Good et. al. in their paper, *The performance of modularity maximization in practical contexts* [58]. This indicates that our implementation of the algorithm is correct and we can also conclude that weighted networks can escape the resolution limit. But it is necessary to inspect the structure of the modules found as we evaluated our algorithm on artificial graphs. So, we must analyze the different communities generated by the community detection algorithm.

3.4 Recommendation systems

Recommendation systems are widely used by retailers to suggest products to the customers. Retailers want to generate personalized coupons for their customers. These coupons contain products that they buy already or are most likely to buy given the (customer's) purchases in the past. Recommending customized coupons to customers has a huge potential of increase in sale for the retailer and thus is considered an important area of research. This not only involves mining past purchases but understanding the behavior of the customer and predicting what products a customer is most likely

to buy.

Recommendation systems are categorized into three categories [5]. First category is one in which the users are suggested items that are selected by similar users (exhibiting similar buying patterns) and is known as *collaborative recommendation*. Second category is one that recommends users items similar to the ones the user preferred in the past and is known as *content-based recommendation*. Third category is one in which the user gets recommended items using additional information about both the customer and the items and is known as *knowledge-based recommendation*.

We build a collaborative recommendation system. We generate clusters of products that are bought together based on the store's customers purchasing data. Once we have the clusters (or communities) of products, we determine a customer's belonging factor to each cluster. For this, we calculate the belonging factor for each customer to each cluster by dividing the number of products that a customer buys that belong to a cluster by the total number of products bought by the customer [5]. The belonging factor will always be between 0 and 1. If there are no products bought from a cluster, belonging factor is set to be zero.

Chapter 4

Processing the Instacart Retail Data

In this chapter, we describe our application processing the Instacart dataset [1]. Our goal is to create a platform, using principles of graph theory, that can be utilized by retailers to understand customer's buying behavior and provide these customers with appropriate coupons. After collecting and understanding the dataset, we preprocess the data to prepare it for loading into the database. The data is loaded as a graph with nodes as the unique attributes and edges as the relationships between the attributes. To achieve our goal, we construct an ad-hoc product network over our model and use clustering algorithm to generate clusters of products such that products within a cluster are much stronger related to each other than with products in another cluster. Then, for recommending products to a customer by means of coupons, we find a customer's "belonging" factor to each cluster. The belonging factor of a customer to a cluster depends upon the ratio of the number of products bought by the customer that belongs to that cluster to the total number of products bought by the customer. Based on this factor, a retailer can recommend products from the clusters. In this section we describe each step in detail.

4.1 Data Set

The data set used in our project is an open-source dataset published by Instacart [1]. Instacart is an online retail service that delivers groceries. Customers select groceries through a web application from various retailers and the groceries are delivered to

them by a personal shopper. The data was published in 2017 for research purposes. The dataset contains over 3 million grocery orders from more than 200,000 customers. The dataset is anonymized and does not contain any customer information. It only contains a unique identifier(id) for each customer. The data is available in multiple comma-separated-value (csv) files. The first file contains product information that includes product id, product name, aisle id and the department id. The aisle id represents the identifier that signifies where the product is placed in the store and department id signifies the category to which the product belongs to.

The second file contains orders and it links each order id with the customer who ordered the same. It also contains the day of the week, hour of day and sequence in which the order was placed. The third file contains the products purchased in each order with the sequence in which the product was added and whether it was reordered by the customer. The fourth and fifth files are the metadata for aisles and departments respectively, containing unique ids and names for each. For each user, there are multiple orders and each order has multiple products. We saw the relational data model of the data in Figure 2.1 of Chapter 2.

4.2 Preprocessing Data

Preprocessing the data is an important phase in analytics before the data is loaded and analyzed. We inspected the data for missing values and random errors such as duplicate identifiers for two or more products and same order identifiers associated with different customers. The product names were also cleaned before loading into the database.

4.3 Data Representation

After cleaning and preprocessing, the data was ready to be loaded into the database. The data was loaded in Neo4j [3] from our application built in Java. The application uses Java Neo4j Native API [21] to upload the data. In order to avoid creating duplicate nodes for the same customer, order and product, an unique constraint is applied on the respective identifiers.

Procedure for creating (: PRODUCT) - [: INORDER] -> (: ORDER) graph:

1. For each Row in Order-Product file
2. Find the Order node with the respective orderid. If not found, create a node with label as Order and property as the orderid.
3. Find the Product node with the respective productid. If not found, create a node with label as Product and property as productid.
4. Then create a [: INORDER] relationship between the Product and the Order.
5. End
6. Return the Graph created.

FIGURE 4.1: Procedure for creating relationship between Orders to Products

Procedure for creating (: CUSTOMER)-[: ORDERED]-(: ORDER) graph:

1. For each Row in Customer-Order file
2. Find the Customer node with the respective customerid. If not found, create a node with label as Customer and property as the customerid.
3. Find the Order node with the respective orderid.
4. If there is no relationship created between the customer and the order, create a [: ORDERED] relationship directed from the customer node to the order node.
5. Find the DayofWeek node with the respective order_dow. If not found, create the node with label as Day and property as the order_dow. Then create a relationship from the Order to the DayofWeek node with relationship [: DAYOFWEEK], if it doesn't exist.
6. Find the HourofDay node with the respective order_hour_of_day. If not found, create the node with the label as Hour and property as the order_hour_of_day. Then create a relationship from the Order to the HourofDay node with relationship [: HOUROFDAY], if it doesn't exist.
7. End
8. Return the Graph created.

FIGURE 4.2: Procedure for Creation of Nodes in Neo4j

The application first loads the orders and products from the Order-Product file. It creates (:*ORDER*) and (:*PRODUCT*) nodes and [:*INORDER*] relationships from products to orders. Then the application loads the customers related to the orders (already loaded in the database) from the customer-order file. It creates a (:*CUSTOMER*) node and [:*ORDERED*] relationship from a customer to an order. It also creates (:*DAY*) and (:*HOUR*) nodes and two relationships, named [:*DAYOFWEEK*] and [:*HOUROFDAY*],

from an order to (:*DAY*) and (:*HOUR*) nodes respectively. The psuedo-code for the creation of relationship between orders to products and customers to orders are presented in Figures 4.1 and 4.2 respectively.

Finally, the metadata for products, aisles and departments are loaded from the respective files. Each product is represented by an unique node in the database and is labeled as (:*PRODUCT*) with product id and product name as its properties. Each aisle and department is also represented by unique nodes and is labeled as (:*AISLE*) and (:*DEPARTMENT*) and aisle id and department id as its properties respectively. There is a [:*ON*] relationship created from (:*PRODUCT*) to (:*AISLE*) and [:*IN*] relationship created from (:*PRODUCT*) to (:*DEPARTMENT*). Figure 4.3 shows the final data model.

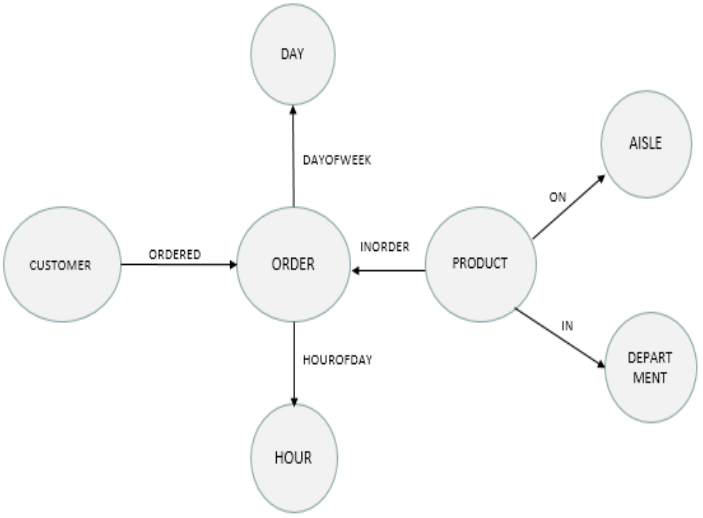

FIGURE 4.3: Data Model Loaded in Neo4j graph database

4.4 Analyzing the Data

After loading the data, we explore the dataset, using Neo4j, for our deeper understanding. We loaded only partial dataset so that it is easy to analyze and work with. In real applications where the data will grow everyday, data centers, with huge storage capacities, can be used for storing and maintaining the dataset. Table 4.1 shows count

of each type of label, named as customers, orders, products, aisles and departments, loaded in the database. We find that on the average 10 products were bought in an order with the maximum being 127 products and minimum being a single product.

TABLE 4.1: Dataset used in the Project

Label	Count of Nodes
Orders	297,453
Customers	126,099
Products	43,179
Aisles	134
Departments	21

The data includes day of the week and time of the day when each order is placed. Figure 4.4 shows the day-of-the-week distribution of when the orders were placed. The orders are shown in the order from the beginning of the week (Monday) to Saturday and Sunday. The days have been represented by numbers, 0 to 6. The data given, by Instacart, did not specify the day indicated by each number. But from the figure, we can see that 0 and 1 are the days when most of the orders are placed and they are most likely to represent the weekends. If 0 and 1 are weekends, we can see that Wednesday, represented by number 4, has the lowest number of orders. Figure 4.5 shows hours distribution of when the orders were placed. The hours are given in 24 hour format and we observe that most of the orders were placed during morning and afternoon hours rather than night hours.

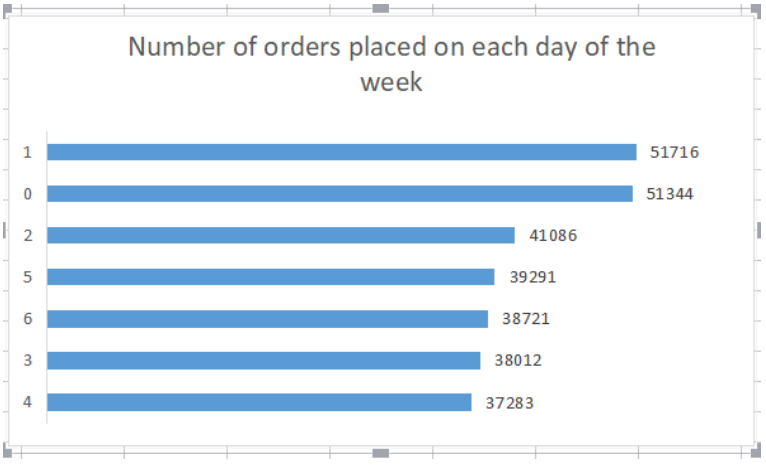

FIGURE 4.4: Number of orders placed on each day of the week

We can also present the day and the time of the orders placed together as in Figure 4.6. That figure shows that the weekend mornings and evenings are the most busy of

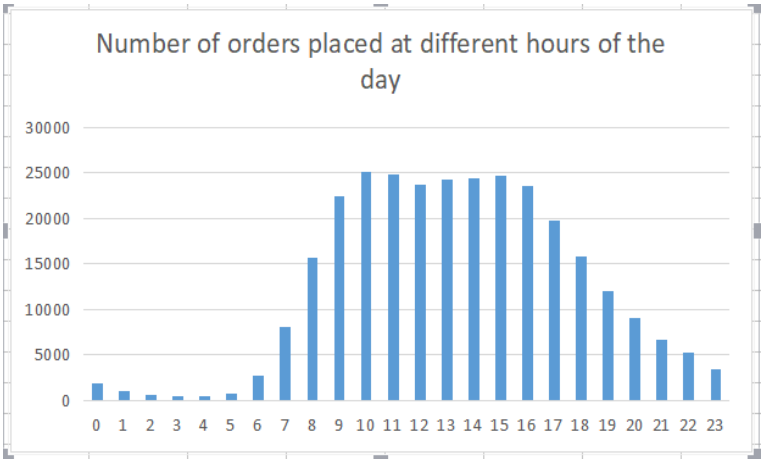

FIGURE 4.5: Number of orders placed at different hours of the day

all the time. This information is useful for the retailers to decide shifts for their staffs and can also help in deciding the hours that are more suitable to restock the inventory.

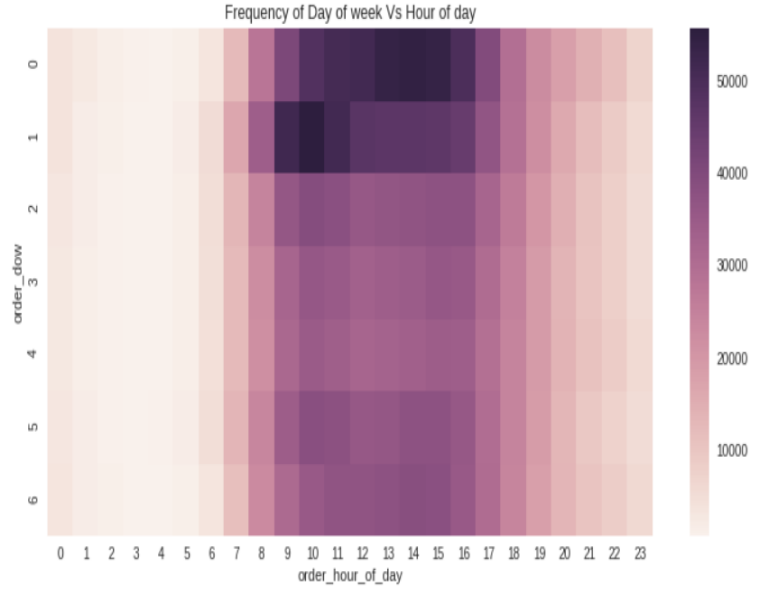

FIGURE 4.6: Number of orders placed at different hours of the day on various days of the week

Next, we looked at the top products, departments and aisles as shown in Figures 4.7, 4.8 and 4.9. We observe that organic products from produce department in fresh fruits and vegetable aisles are the most bought products by the customers of Instacart.

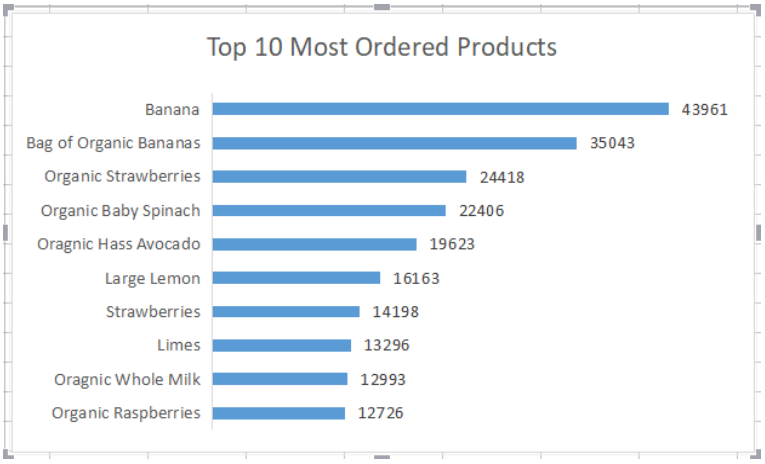

FIGURE 4.7: Top 10 Most Ordered Products

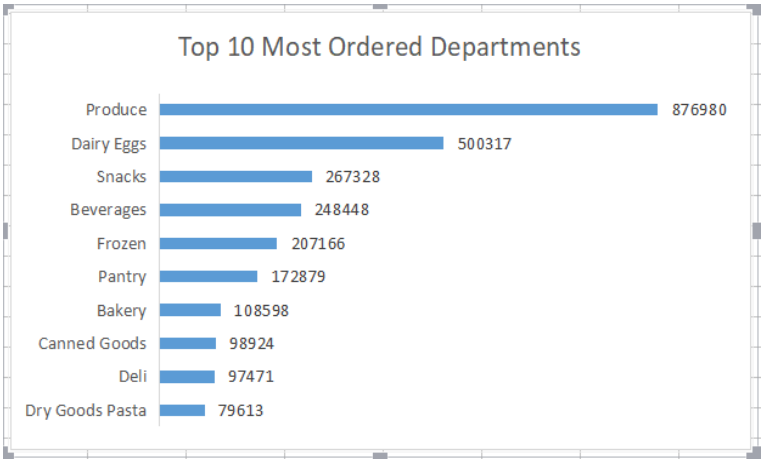

FIGURE 4.8: Top 10 Departments

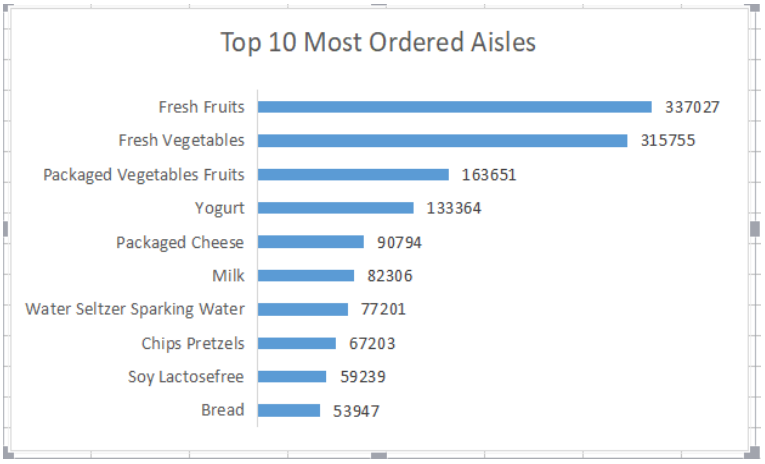

FIGURE 4.9: Top 10 Aisles

4.5 Product-Product Network

To produce our recommendation system, we build a product-product network on top of the data loaded in the database as described above. The transaction data directly relates a customer to the orders placed by them and the products bought in each order. The collected data does not directly create relationships between products. In order to create clusters of products, we have to create a network of products where each product will be associated with other products based on some measure.

To build the product-product network, we create a relationship (using an edge) between every pair of product, if they are bought together in any order. The relationship that is created between two products is termed [:*BOUGHTWITH*] with a property named as weight. The weight on the relationship describes the number of orders in which both the products were bought together. The assumption is that products that are frequently bought together in any same order shows higher association with each other. Weight is usually a number starting from 1. There are no relationships of weight 0. Figure 4.10 contains the procedure used for creating the product-product network. Table 4.2 shows the number of nodes and edges created in the product-product network.

Procedure for (: PRODUCT)-[: BOUGHTWITH] -> (: PRODUCT) graph:

1. For each two products that is in the same order
2. If there is no relationship created between two products
   a. Create a Relationship [: BOUGHTWITH] between the two products with property Weight of value 1;
3. Else add 1 to the value of weight of the Relationship.
4. End
5. Return the Graph created.

FIGURE 4.10: Procedure for creating the product-product network

After the creation of ORDERED, INORDER and BOUGHTWITH relationships, the product-order network looks like Figure 4.11. For building our recommendation system, we only take the Product-Boughtwith-Product relationships, that we name - Product network.

TABLE 4.2: Number of Nodes and Edges in Product Network

Nodes	Edges/BoughtWith Relationships
43,179	9,215,696

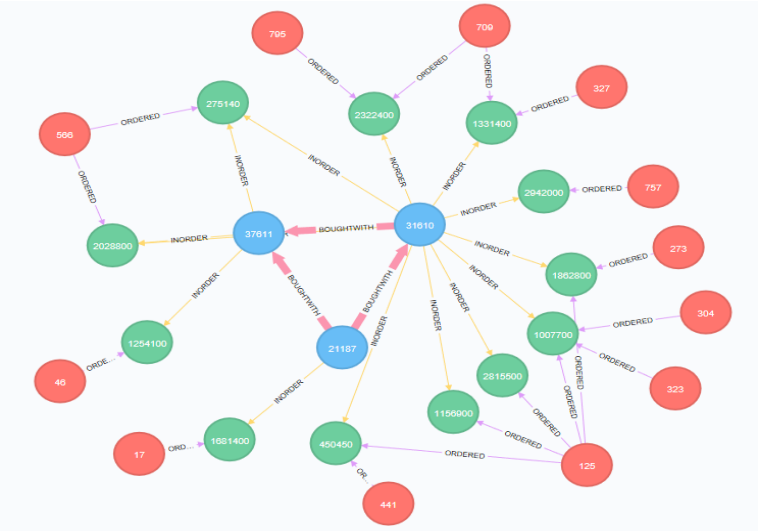

FIGURE 4.11: Relationships among Customers, Orders and Products

4.6 Community Detection

Once the product network is built, we apply the Louvain community detection algorithm [3] to generate clusters of products that will group products that are often bought together with each other. We use Neo4j's implementation of Louvain algorithm [21] because it is faster on larger dataset than our own implementation. The algorithm is applied to the product network generated from the data. The algorithm returns 48 communities and the largest community is of size almost 43,000. The algorithm grouped most of the products together rather than creating relatively smaller meaningful clusters. This is because the graph is highly connected and the same is also justified by looking at the ratio of number of nodes and edges provided in Table

4.2. As the Instacart data is real transaction data, we can assume that real life transaction data from similar retailers can face similar problem. So, we propose three models for clustering similar transactional data and evaluate their results.

To build clusters that we can analyze and use, we looked at the weight distribution of the relationships in the product network. The result is shown in Table 4.3. We found that 97 percent of edges have weight less than 10. This means 97 percent of pairs of products has been bought together in less than 10 orders. There were 0.003 percent of edges with weight more than 50. There are also edges whose weight is bigger than 1000, showing higher associated degree between certain pair of products. Table 4.4 shows some of the such highly connected products.

Table 4.3: Weight Distribution in the Product Network

Weight Range	**Count of relationships**
0-9	8,958,407
10-19	155,799
20-29	42,834
30-39	19,086
40-49	10,668
50-99	18,369
100-199	6,903
200-999	3,418
1000- 5700	212

Table 4.4: Some pairs of highly connected products

Products bought together in an order	**Number of orders**
"Banana", "Organic Strawberries"	5206
"Organic Baby Spinach", "Bag of Organic Bananas"	4623
"Organic Hass Avocado", "Organic Strawberries"	3791
"Bag of Organic Bananas", "Organic Raspberries"	3656
"Organic Baby Spinach", "Organic Avocado"	2881

We now propose our first model, based on the weight distribution. We recognize that the edges must have weights bigger than a certain threshold to be considered in the community detection process. This number depends upon the data and the weight distribution. Based on Table 4.3, we decided that edges with weights less than 10 are not significant, given the number of orders in our dataset. So, we applied the algorithm on the network filtering out the edges whose weight is less than 10. The algorithm produces 34725 communities and the top five communities with respect to

their sizes are presented in Table 4.5. The size of the communities are not fixed and (often) change in every run. This is due to the design of Louvain algorithm. Louvain algorithm is dependent on the order in which the nodes are considered in each run, so we ran our models several times. But we also found that the proportion of the sizes does not vary much. We can see that the algorithm performed better than before by using filtering. There is one large community followed by relatively smaller communities.

TABLE 4.5: Community Detection on the Network with weight > 10

Community	Community Size
1	8343
2	51
3	19
4	10
5	4

To compare the communities so obtained, we looked at the individual departments. We saw that the first two communities had the combination of departments like dairy eggs, produce, pantry, meat, seafood, bakery, snacks etc. But the remaining communities are from within the same department. The third community is from pets departments and contains only cat food, fourth community is from baby department and fifth is from snacks department. This shows the presence of tightly-knit communities in the data. Figure 4.12 shows the last three communities from Table 4.5.

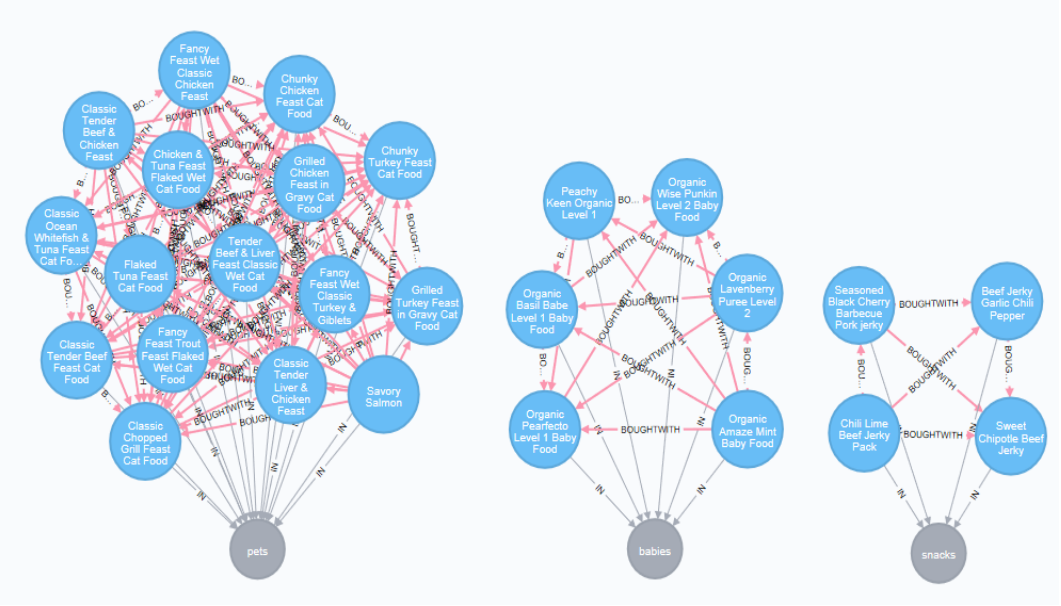

FIGURE 4.12: Some of the communities obtained from first model

As our goal is to create recommendation system, we want to identify more tightly-knit clusters available in the data. We like to avoid a large community, as we found in our first model, as it is difficult to analyze and efficiently use such communities. We analyze our data further and try to achieve better clusters. We looked at some of the products to see their weight distribution. Table 4.6 shows the relationships of organic strawberries with five other products. We can see that the weight distribution of organic strawberries are higher with bag of organic bananas, limes and organic kiwi and less with sweet onions and soft potato bread. This means that organic strawberries shows stronger probability of being bought together with first three products than with the last two. So, we should consider the first three products for organic strawberries rather than all 5. But there is no "correct" number of orders that represents higher connectivity between two products. This number is subjective and depends upon the data. This number, however, can be found by analyzing the data.

TABLE 4.6: Weight distribution for "Organic Strawberries"

Products	**No. of Orders**
"Bag of Organic Bananas"	5661
"Limes"	1743
"Organic Kiwi"	1126
"Sweet Onions"	71
"Soft Potato Bread"	64

Based on the above idea, we implemented our second model of clustering. We create a subgraph on top of product network. This graph filters out the lower weighted edges attached to each product and only considers higher weight edges. This means that if a product is associated with N other products with various weights, then we consider only top M highest weighted edges (where $M \leq N$ and N is chosen based on the analysis on the data). We consider M as the threshold of the subgraph.

We ran several experiments with different thresholds. Table 4.7 shows the sizes of the 10 communities out of 37,602 communities formed in the subgraph created that only considered the edges whose weights were more than 10 and two highest weighted edges connected to a product.

The clustering results (shown in Table 4.7) show better distribution of clusters than our first model. Unlike model 1 where we got one big cluster, this model produces

TABLE 4.7: Community detection on the network with weight > 10 and each product connected to two other products with highest weights.

Community	Community Size
1	2376
2	1763
3	752
4	44
5	18
6	18
7	16
8	15
9	14
10	14

communities of moderate sizes. To evaluate this model, we looked at the products in these clusters. We found that the products exhibit strong similarity behavior and give better clusters that can be used to recommend products to customers. Figure 4.13 shows the list of products present in cluster 5 of Table 4.7 and Figure 4.14 shows the list of product present in cluster 7 of Table 4.7. We can see that cluster 5 contains products that are mostly combination of products from snacks and beverages department that might be related to purchases done for a party as it contains products like nuggets, cookie tray, orange soda, juices etc. On the other hand cluster 7 contains products that are mostly unsweetened beverages and might be related to customers who are diabetic or are health conscious. We believe that this model can be used for recommending products.

In our third model, we created communities within departments. As each department has high number of products, we looked at the communities formed within a department to study if products can be recommended based on the categories of products bought by a customer. We experimented on some of the departments and analyzed the results. Table 4.8 shows 5 communities out of 460 communities generated in Babies department, that has 972 products with 25,623 boughtwith relationships among products. Figure 4.15 shows a pictorial representation of these communities, differentiated by various colors, in Babies department obtained using Gephi tool [56]. The size of the node is dependent on the between-centrality and width of the edge is dependent on the weight.

productlist	department
"Assorted Nuggets"	"snacks"
"Drumsticks"	"frozen"
"Cookie Tray"	"bakery"
"Fruit Snacks"	"snacks"
"Chocolate Sandwich Cookies"	"snacks"
"Sparkling Water Bottles"	"beverages"
"Movie Theater Butter Microwave Popcorn"	"snacks"
"Steak Strips"	"snacks"
"Orange Soda"	"beverages"
"100% Orange Juice"	"beverages"
"Milk Chocolate Covered Raisins"	"snacks"
"Sugar Free Energy Drink"	"beverages"

FIGURE 4.13: List of Products in cluster 5 formed on the subgraph of the Product Network

productlist	department
"Unsweetened Cucumber Water"	"beverages"
"Unsweetened Pear Essence Water"	"beverages"
"Unsweetened Honeydew Essence Water"	"beverages"
"Unsweet Pineapple Water"	"beverages"
"Water Unsweet Blood Orange"	"beverages"
"Mango Grapefruit Water"	"beverages"
"Unsweetened Pomegranate Essence Water"	"beverages"
"Water Unsweet Crisp Apple"	"beverages"
"Unsweetened Strawberry Kiwi Water"	"beverages"
"Unsweet Peach Water"	"beverages"
"Raspberry Essence Water"	"beverages"
"Unsweetened Blackberry Water"	"beverages"
"Unsweetened Watermelon Water"	"beverages"

FIGURE 4.14: List of Products in cluster 7 formed on the subgraph of the Product Network

TABLE 4.8: Community Detection in "Babies" department on the Product Network

Community	Community Size
1	29
2	7
3	7
4	6
5	5

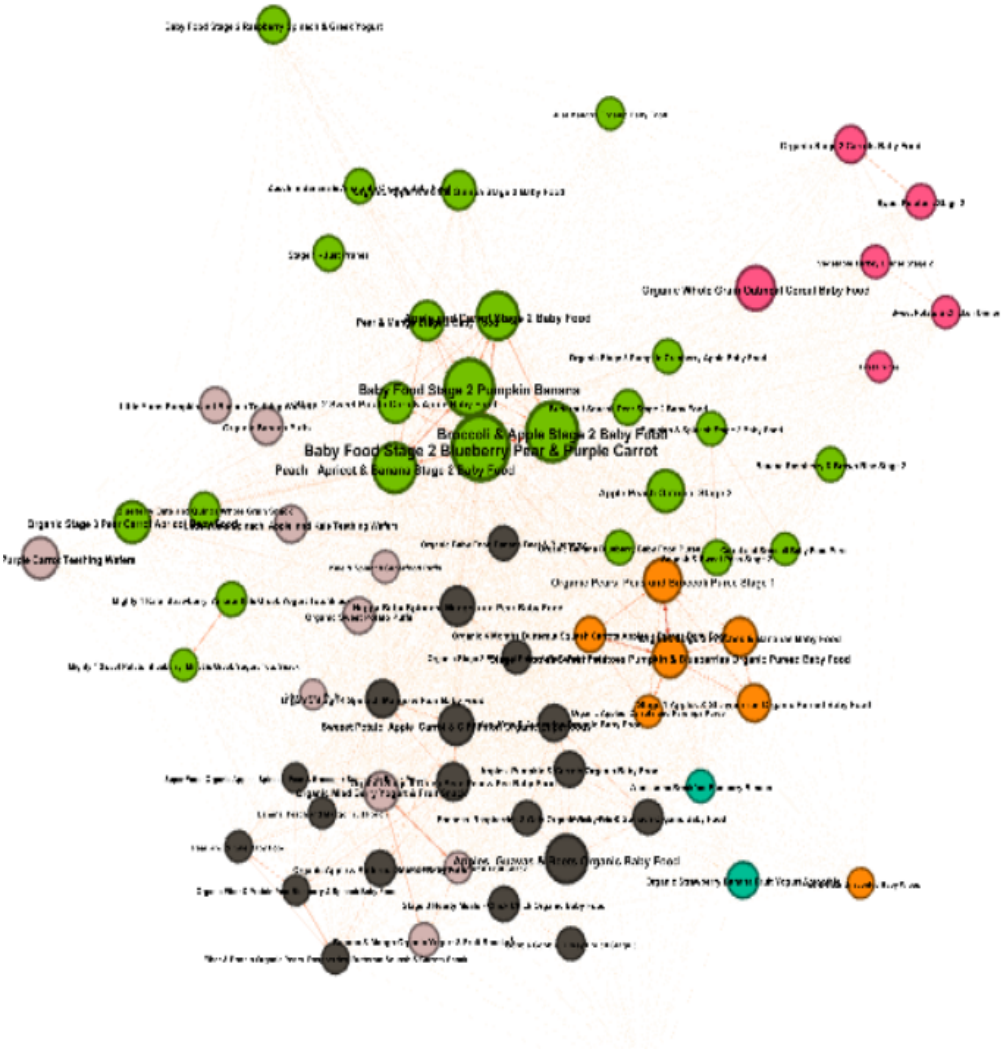

FIGURE 4.15: Communities in "Babies" department using Gephi

Table 4.9 shows the details of the products available in three of the communities generated within personal care and international departments. This shows that strong communities are present within departments as well.

TABLE 4.9: Products in communities formed within "Personal Care" and "International" departments on the Product Network

Department	**Products in 1st Community**	**Products in 2nd Community**	**Products in 3rd Community**
personal care	Clarifying Shampoo, Conditioner Daily Nourishing, Moroccan Argan Oil + Argan Stem Cell Triple Moisture Conditioner, Hair Shampoos	All In One Coconut Almond Packet, One Plant-Based Chocolate Flavor Nutritional Shake Drink Mi, All-In-One French Vanilla Nutritional Shake Sachet	Bluebell Hand Soap, Rosemary Hand Soap, Olive Oil Aloe Vera Hand Soap, Honeysuckle Hand Soap, Liquid Hand Soap Refill Lavender Scent, Lavender Hand Soap, Clean Day Basil Hand Soap
international	Asian Vegetable Ramen, Seaweed Ramen, Mushroom Ramen, Organic Lemongrass Ginger Ramen, Tofu Miso Ramen, Garlic Pepper Ramen	Wasabi Roasted Seaweed Snacks, Roasted Sesame Seaweed Snacks, Organic Roasted Seaweed Snacks Teriyaki, Organic Sea Salt Roasted Seaweed Snacks	Madras Lentils Indian Cuisine, Indian Cuisine Bengal Lentils, Channa Masala Indian Cuisine

4.7 Web Application For Recommendation System

To build the recommendation system, we create an application that can be used to visualize and analyze the data and then choose a model to recommend products to customers. We built the application in Java using Neo4j API [21]. We have also used JavaScripts provided by Google Charts [60] and jQuery [61] to make the application more interactive and user-friendly. There are two main function of the application, one is viewing the data at different levels and the other is analyzing the clustering results and recommending products to the customers.

Figure 4.16 shows a screen shot of the view data page in the application. It provides various links that can be used to view the data at different levels. Figure 4.17 shows a screen-shot of one of the pages in the application where a user can view the various departments in the inventory. The departments are sorted by the number of times products are ordered from each of them.

After viewing the data, users can create communities, analyze the results and choose a model to recommend products to a customer. We have implemented all three models for generating clusters, that we discussed in the previous section. Figure 4.18 shows a screen shot of the home page for community detection in the application where a user can browse through the three models and analyze the results.

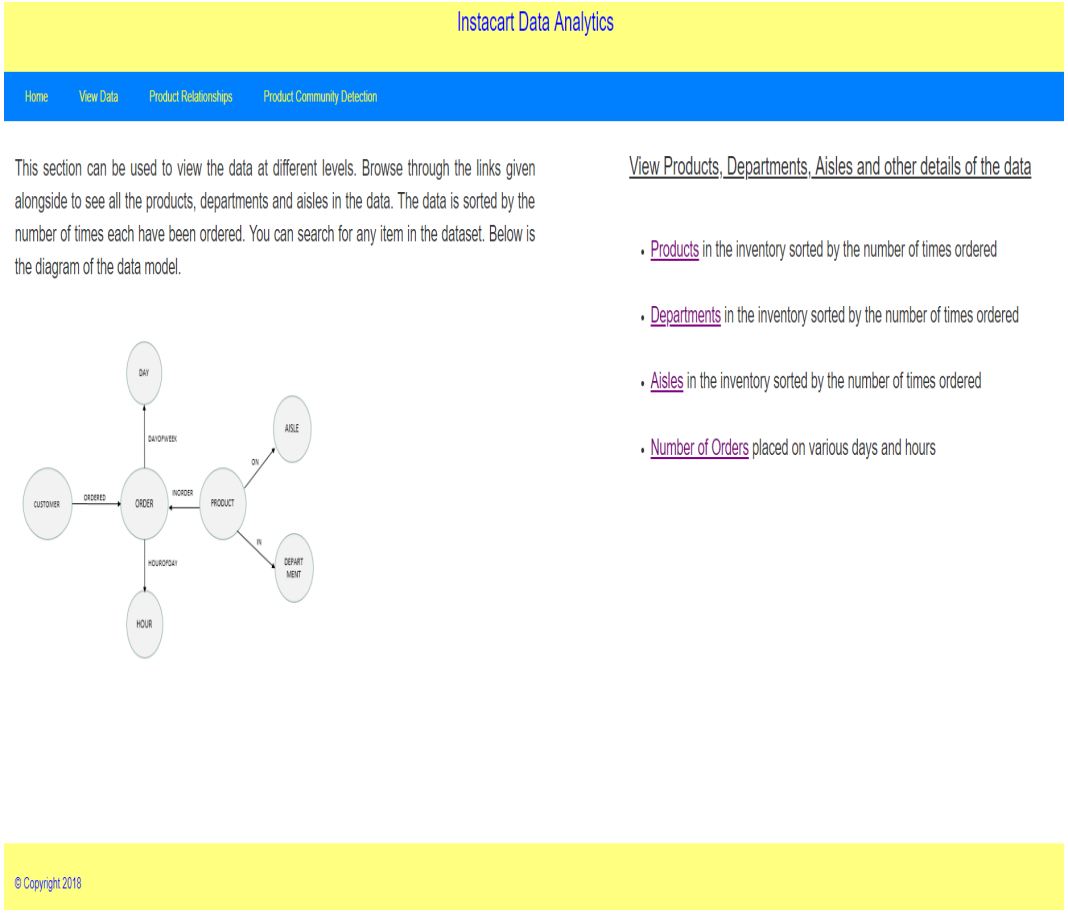

FIGURE 4.16: View Data Page from the application

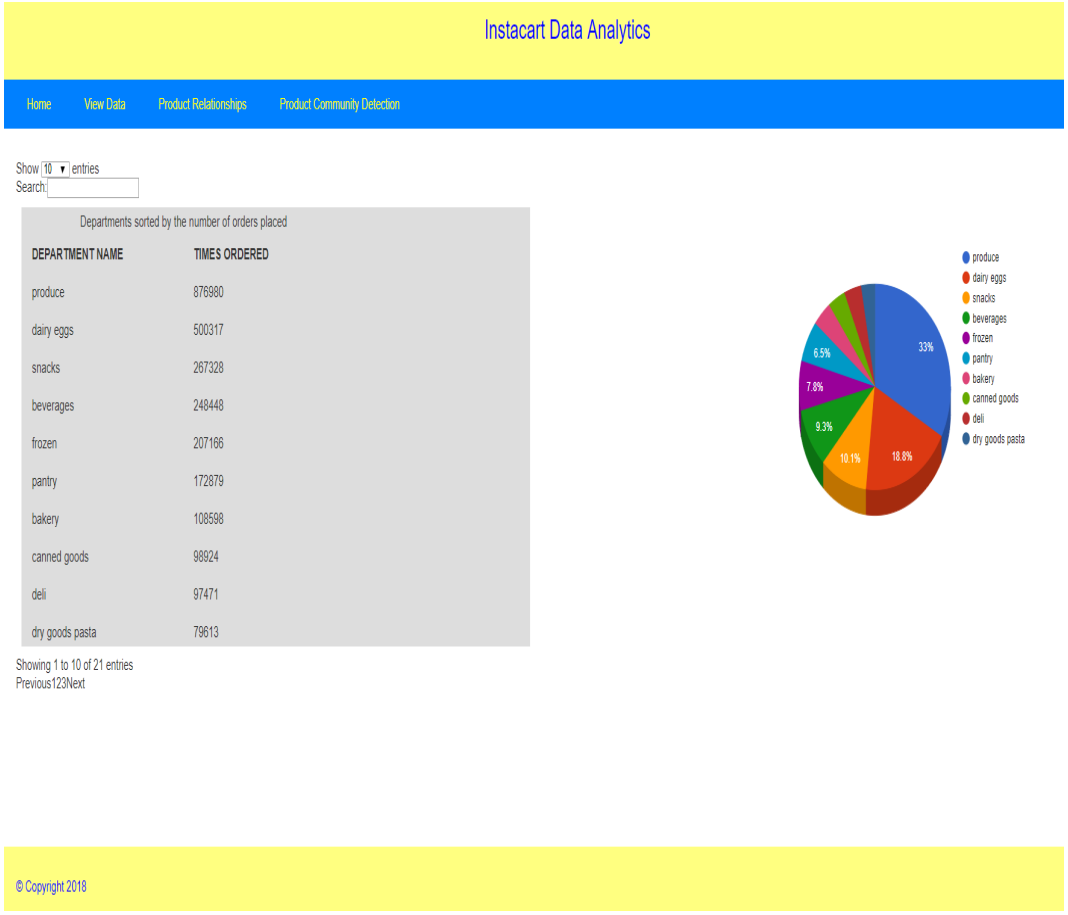

FIGURE 4.17: Departments Page from the application

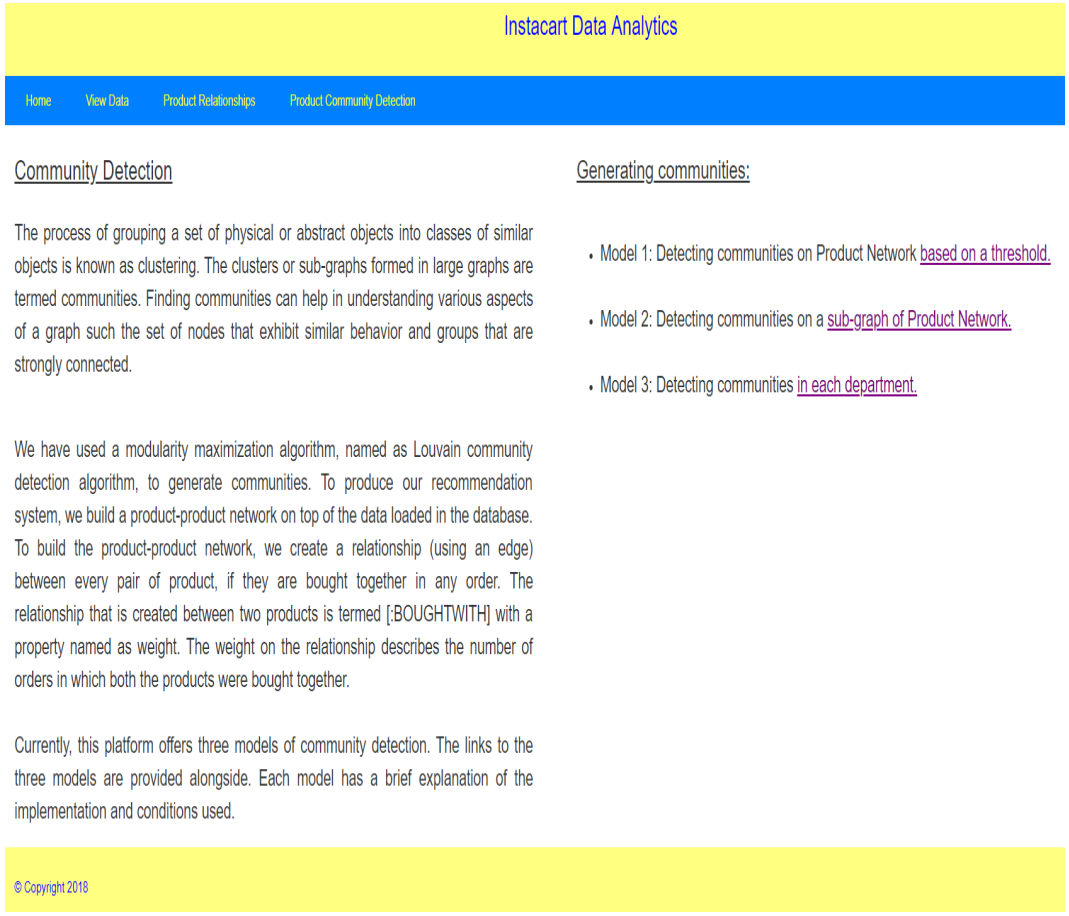

FIGURE 4.18: Community Detection Home Page from the application

The functionality of first two models are quite similar. When the user clicks the first or the second model, the application creates communities in Neo4j graph database and presents the results on the screen. The user can analyze the results by looking up the products in each community along with their respective departments and aisles. A user can also analyze the results of recommending products to a customer. This can be done by providing user's id and the application will calculate the belonging factor of that customer to each cluster created by the model. Then the application chooses the first two clusters that have the highest belonging factor and recommends products from those clusters. Figure 4.19 shows the page in our application when a user chooses model 2. As we can see, there is an option to view products in each community and second option is to recommend products to a customer.

Figure 4.20 shows the page when a user can see the products in each community. This page can be used to analyze the results and determine if the model can be used for recommending products to a customer. Figure 4.21 shows the page where the list of recommended products to a customer and the list of the products that the customer has bought in past are displayed. This page can also be analyzing by analysts if the model can be used for recommending products to a customer.

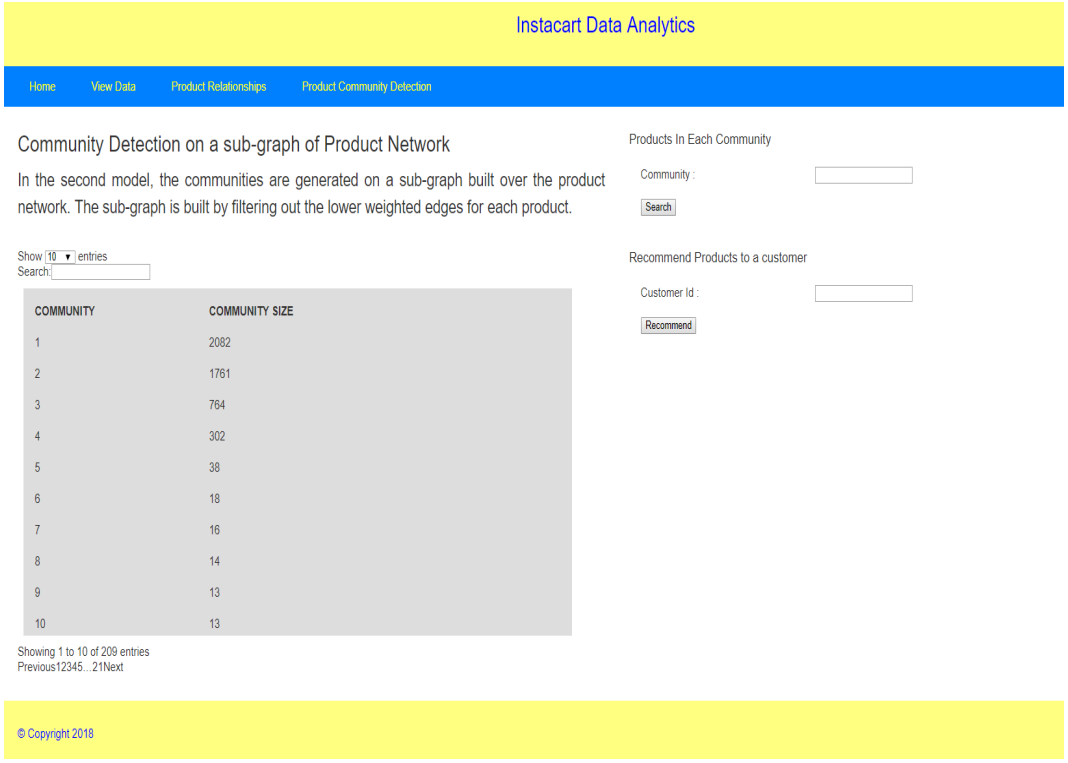

FIGURE 4.19: Screen shot of Recommendation Based On Sub-Graph of the Product Network

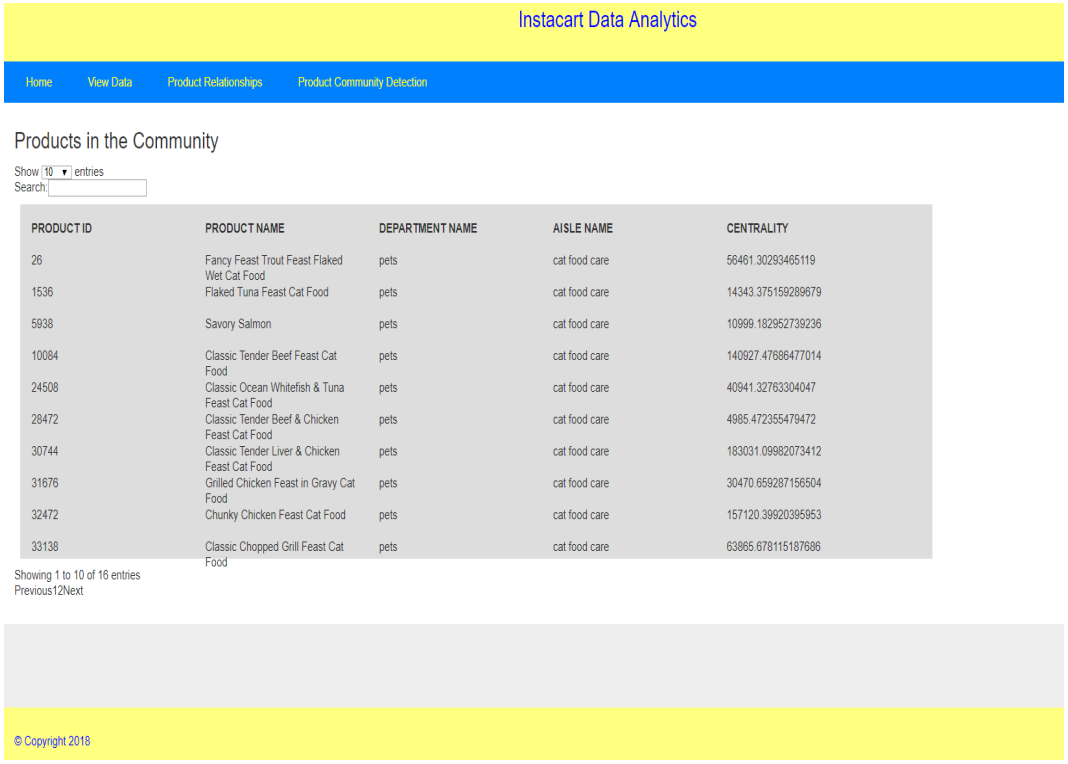

FIGURE 4.20: Products in a Community Page from the application

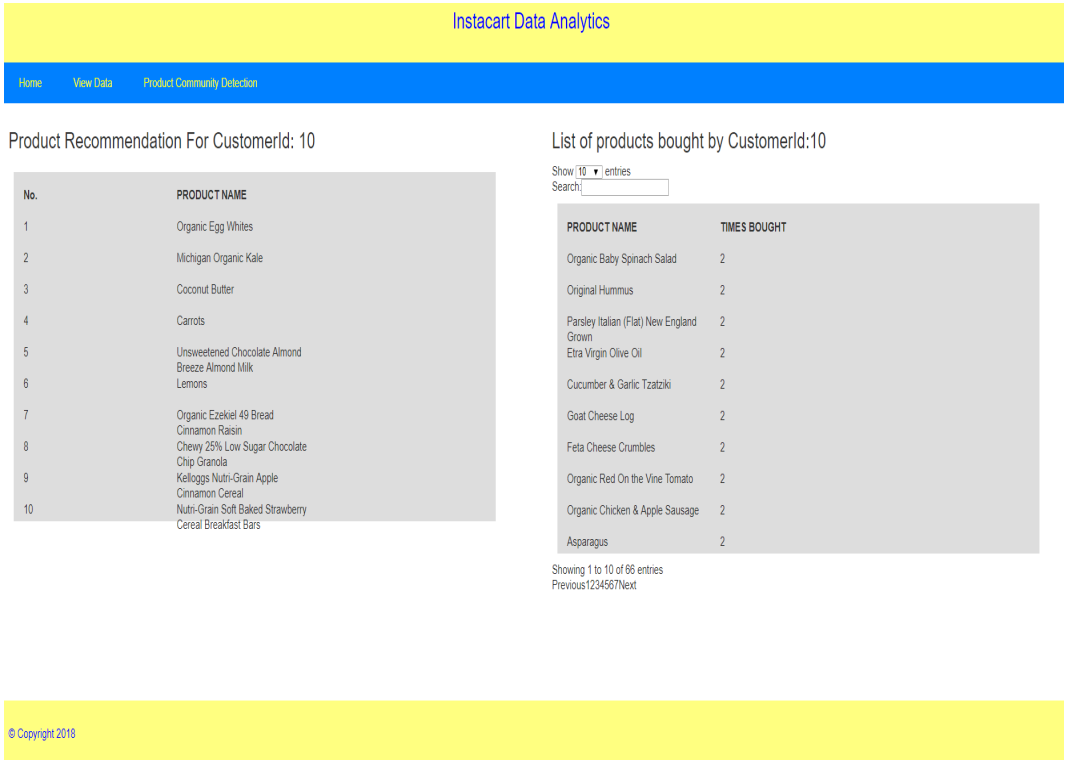

FIGURE 4.21: Example of Product Recommendation For Customer "Id10" in Model 2

In the third model, a user has to enter the customer's id and the application fetches and displays the departments from which the customer has ordered products and their respective counts. The user can then give the customer's id and select any two departments from which they want to recommend products. Figure 4.22 shows the page when a user selects the third model and looks for departments for customer Id10.

The application is built for providing an user-friendly interface for the analysts, without worrying about the back-end processing. As the business strategies need constant change, we have kept certain conditions flexible that can be easily changed. In our model, we have selected the first two clusters with highest belonging factor but this condition can be changed depending on the business needs. Our application demonstrates that market basket analysis can be done using a graph database.

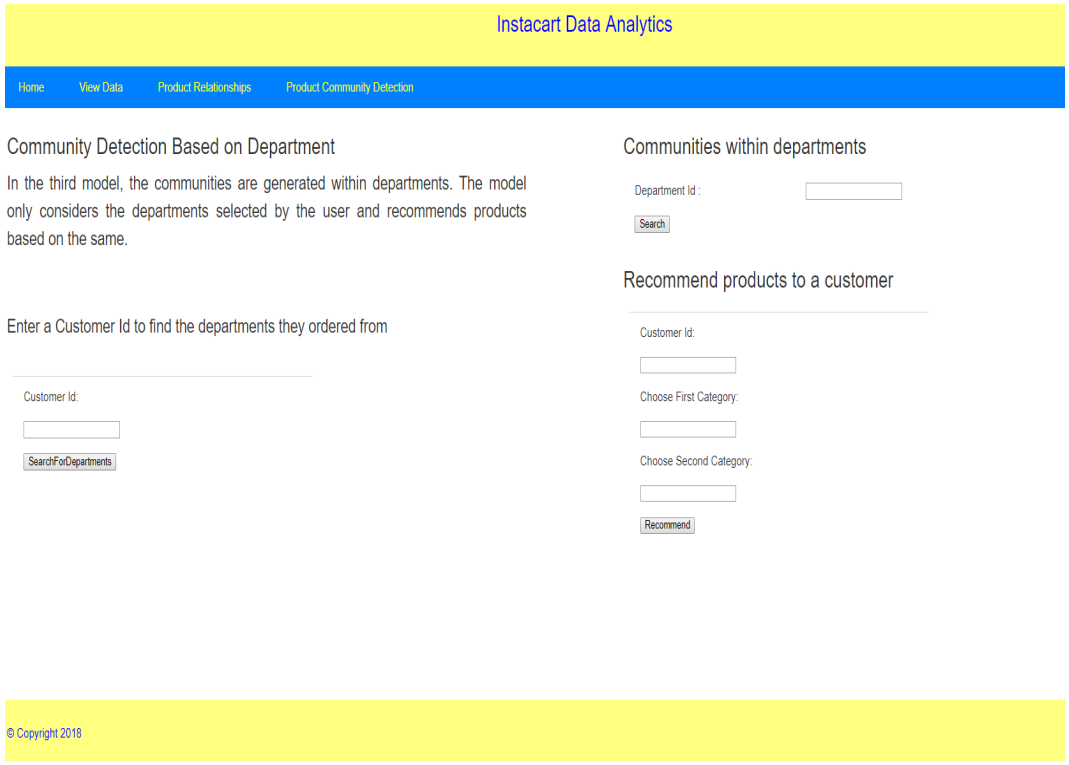

FIGURE 4.22: Recommending Products within Departments Page from the application

Figure 4.23 shows the recommended products from deli and dairy eggs departments.

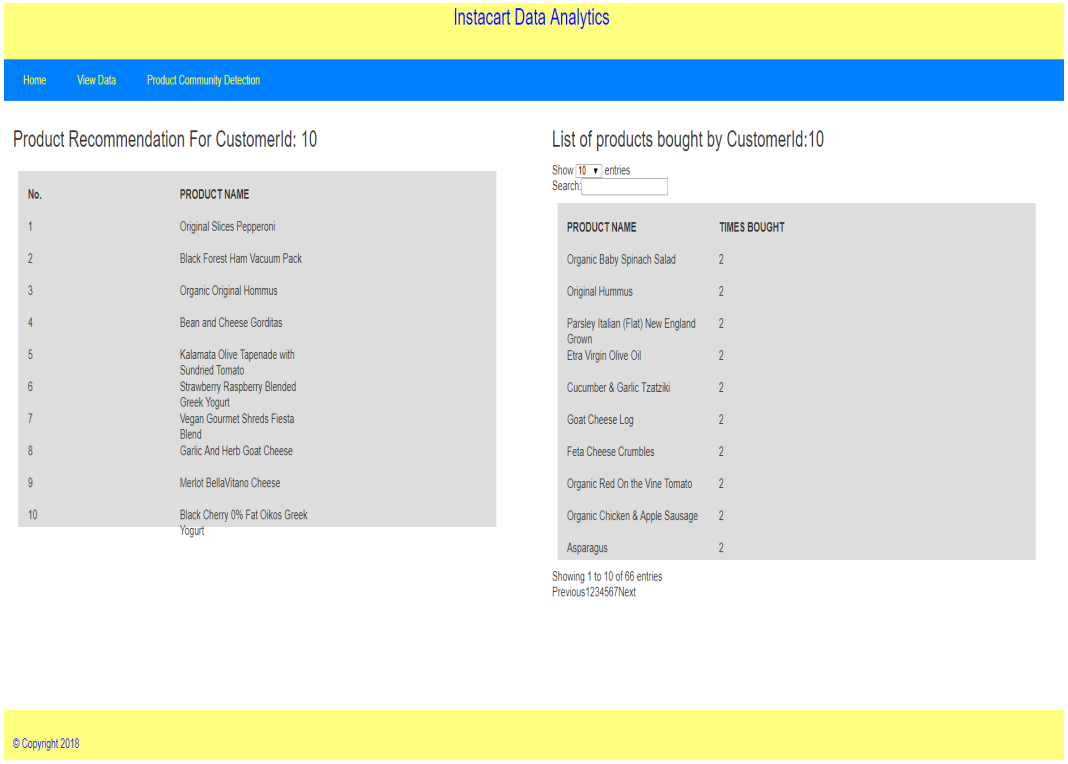

FIGURE 4.23: Recommendation based on Departments

Chapter 5

Performance of Neo4j

In this chapter, we discuss the performance of Neo4j. Overall, the combination of our chosen data model and Neo4j proved to perform well when querying the dataset and performing exploratory data analysis. Our experiments are not performance benchmark for Neo4j but they do reflect the general performance of Neo4j. However, we discuss here some of the performance related issues.

We found that Neo4j needs much RAM for some operations such as creating ad-hoc subgraphs over large networks. It is advisable to know the size of the graph beforehand in order to determine how much RAM needs to be allocated to the heap. This is because Neo4j does not include a native load balancer and so it relies on the load-balancing capabilities of the network infrastructure to help maximise throughput and reduce latency.

We also observed that Neo4j's performance depends upon how the data is modelled. We had to model our data multiple times before reaching our final design. This is due to the fact that Neo4j organizes properties, assigned to nodes, as Binary Large Object (BLOB) [21]. So, properties are stored into one file, and they are present all over the file. This does not affect small datasets, but for large datasets, it becomes extremely slow to perform certain operations such as creating a sub-graph over the dataset and querying properties of the nodes. Thus, it is a good practice to avoid creating too many properties linked to a node but, rather, create nodes for these properties.

In our final data model we found that Neo4j easily dealt with connected data to support efficient node look-up. Cypher query language facilitates creating indexes for labels that promotes efficient look-up. Indexes and unique constraints also helps in optimizing the process of finding specific nodes.

We present the time taken to run certain queries in Neo4j in Table 5.1. We observed that Neo4j loaded large datasets with relationships in a reasonable time. The time taken to query details of products bought by a customer was fast. The most expensive operation was creating a sub-graph over the original graph.

Overall, we found that Neo4j performed very well while handling large dataset. It can be used for processing large and growing interconnected datasets like retail data. Neo4j graph database is relatively a newer database in comparison with some relational databases and issues related to memory should be addressed for efficient processing of large datasets.

TABLE 5.1: Time taken to run queries in Neo4j (in seconds)

Query Description	**Time Taken by Neo4j**
Loading almost 43,000 nodes	1.87
Loading 340,632 nodes (*ORDERS* and *PRODUCTS*) and 3,000,000 (:*PRODUCTS*)-[:*INORDER*]->(:*ORDER*) relationships	40.55
Loading 126,099 nodes (*CUSTOMERS*) and 297,453 (:*CUSTOMER*)-[:*ORDERED*]->(:*ORDER*) relationships	23.22
Creating product-product network, with over 43,000 nodes and over 9 million edges, with weights as property	2400
Querying all the products bought by a customer, along with the departments and aisles they are in (time taken is the average time taken obtained by querying for some customers)	0.049
Louvain community detection over the entire product-product network	75

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We presented a process for retail data analytics using graph representations. Retail is one of the domain that collects huge amount of data everyday. The data is interconnected and traditional methods are not useful anymore for the identification of existing relationships between products. An important contribution of our work was building a platform for retailers that can be useful for getting insights into the data that can help them make decisions benefited for their business and quick at the same time. The datasets used in our work was a real transaction data from Instacart [1]. We found that as graphs are very intuitive and easy to work on for high-dimensional data, a graph database can be very effective in such scenario. Neo4j graph database was very efficient in handling huge datasets. Our final dataset contained nodes and edges. We reduced the dataset in order to analyze the data easily and not because of Neo4j's computation power. It was difficult to learn Cypher in the beginning but once we understood its concepts, it became a user-friendly platform.

We used Louvain algorithm [3] for detecting communities in retail transactional data and recommending products to customers. It is a simple heuristic method based on modularity optimization that extracts hierarchical community structure of large networks. We found that community detection algorithms do not perform well in a highly connected graph as most of the nodes are densely connected and this prevents from

obtaining smaller tightly-knit communities. So, we proposed three models for detecting communities in such large dataset. Louvain algorithm has found applications in many areas and we found that it yields good results in retail data as well. The aggregated graphs are clear and show distinct communities. The Neo4j's implementation of Louvain algorithm is fast and generates relationships that are easy to understand and to use in applications.

6.2 Future Work

As retail data analytics is aimed at improving sales for the retailers, it is very important to review the sales of the products that are being recommended to the products. This area is a dynamic field that needs constant improvement. The impacts of the methods applied must be frequently reevaluated. We could also use the temporal aspects of the transactions for generating more stream-lined communities. We used Louvain clustering algorithm to generate non-overlapping clusters of products. We should also evaluate the impact of an overlapping community clustering algorithm and compare the impacts of both methods. We used one dataset in our work, in future, we can generate clusters in datasets from different retailers and compare, analyze and improve the process.

We can also use customer's data to improve the recommendation system. In real projects, we could use order's time-stamp, customer's demographics and geographic details to further improve the recommendation system. Another future work could be to deploy the application on a cloud platform, like Amazon Web Services. The platform can then be used as a single platform for stores at different locations. This can help the retailers to compare results among different stores, between customers in different demographic groups, between different days of the week, different seasons of the year, etc.

References

[1] Instacart Dataset: *https://tech.instacart.com/3-million-instacart-orders-open-sourced-d40d29ead6f2/.*

[2] *https://neo4j.com/.*

[3] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte and Etienne Lefebvre *Fast unfolding of communities in large networks (2008).*

[4] Neo4j Algorithms, *https://neo4j-contrib.github.io/neo4j-graph-algorithms/.*

[5] Ivan F.Videla-Cavieres, Sebastián A. Ríos *Characteriation and completation of the customer data from a retail company using graph mining techniques (2014).*

[6] Georgios Drakopoulos and Andreas Kanavos *Fuzzy Graph Community Detection in Neo4j (2016).*

[7] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C.Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski, Google, Inc. *Pregel: A System for Large-Scale Graph Processing (2010).*

[8] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin and Joseph Hellerstein *GraphLab: A New Framework For Parallel Machine Learning (2010).*

[9] Colin Shearer *The CRISP-DM Model: The New Blueprint for Data Mining (2000).*

[10] Alexandros Labrinidi and H. V. Jagadish, *Challenges and Opportunities with Big Data, Proceedings of the VLDB Endowment VLDB Endowment Hompage archive Volume 5 Issue 12, August 2012.*

[11] Thomas H. Davenport *Enterprise Analytics, Optimize Performance, Process, and Decisions Through Big Data.* International Institute For Analytics.

[12] Veronika Abramova, Jorge Bernardino and Pedro Furtado *Experimental Eavaluation of NoSQL Databases (2014).*

[13] Gerard George, Ernst C. Osinga, Dovev Lavie and Brent A. Scot *Big Data And Data Science Methods For Management Research* Academy of Management Journal, 2016, Vol. 59, No. 5, 1493–1507. http://dx.doi.org/10.5465/amj.2016.4005.

[14] McAfee, A., Brynjolfsson, E. 2012. *Big data: The management revolution*, Harvard Business Review, 90: 61–67.

[15] Santo Fortunato, Complex Networks and Systems Lagrange Laboratory, ISI Foundation, ITALY, *Community detection in graphs (2009).*

[16] Scott, J., *Social Network Analysis: A Handbook (2000),* SAGE Publications, London, UK.

[17] Michelle Girvan and M. E. J. Newman *Community structure in social and biological networks ().*

[18] Paul Erdős and Alfréd Rényi, *On Random graphs*, in Publ. Math. Debrecen 6, p. 290–297.

[19] Jiawei Han, Micheline Kamber and Jian Pei. *Data Mining Concepts and Techniques (2000)*, Third Edition, The Morgan Kaufmann Series in Data Management Systems.

[20] Sami Ayramo and Tommi Karkkainen *Introduction to partitioning-based clustering methods with a robust example,* Reports of the Department of Mathematical Information Technology Series C. Software and Computational Engineering No. C. 1/2006.

[21] Neo4j Developer Manual, *https://neo4j.com/docs/developer-manual/*

[22] Elbow method (clustering), *https://en.wikipedia.org/wiki/Elbow method (clustering)*

[23] Analytics, *https://en.wikipedia.org/wiki/Analytics*

[24] Neo4j Gremlin Plugin, *http://neo4j-contrib.github.io/gremlin-plugin/*

[25] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen and Dawn Wilkins *A comparison of a graph database and a relational database: a data provenance perspective (2010).*

[26] E. F. Codd, IBM Research Laboratory, San Jose, California. *A Relational Model of Data for Large Shared Data Banks, Communications of the ACM CACM Homepage archive, Volume 13 Issue 6, June 1970, Pages 377-387.*

[27] M. E. J. Newman, Department of Physics and Center for the Study of Complex Systems, University of Michigan, Ann Arbor, MI *Fast Algorithm for detecting community structure in networks (2003).*

[28] M. E. J. Newman and M.Girvan, *Finding and evaluating community structure in networks (2003).*

[29] M. E. J. Newman and M.Girvan, *Modularity and community structure in networks (2006).*

[30] Ulrich Elsner, *Graph Partitioning: A Survey. Chemnitz, Germany: Technische Universität Chemnitz; 1997. Technical Report 97-27.*

[31] Per-Olof Fjällström, Department of Computer and Information Science Linkoping University, Sweden, *Algorithms for Graph Partitioning A Survey (1998).*

[32] Stanley Wasserman and Katherine Faust, *Social Network Analysis-Theory and Applications. (Structural Analysis in the Social Sciences), Cambridge, U.K.: Cambridge University Press, 1994.*

[33] Christian Schulz, Karlsruhe Institute of Technology, *Graph Partitioning and Graph Clustering in Theory and Practice (2016).*

[34] Andrea Lancichinetti and Santo Fortunato, *Community detection algorithms: a comparative analysis (2010).*

[35] Edgar F. Codd, *Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate, Codd Associates, (1993).*

[36] SNAP System, by Jure Leskovec, *http://snap.stanford.edu/snap/*

[37] Transportation Networks, National Transportation Atlas Database, Bureau of Transportation Statistics: *https://www.bts.gov/geospatial/national-transportation-atlas-database*

[38] Reguly T, Breitkreutz A, Boucher L, Breitkreutz BJ, Hon GC, Myers CL, Parsons A, Friesen H, Oughtred R, Tong A, Stark C, Ho Y, Botstein D, Andrews B2,6, Boone C, Troyanskya OG, Ideker T, Dolinski K, Batada NN and Tyers M, *Comprehensive curation and analysis of global interaction networks in Saccharomyces cerevisiae (2006).*

[39] Merrill, Jacqueline, Suzanne Bakken, Maxine Rockoff, Kristine Gebbie and Kathleen Carley, *Description of a method to support public health information management: Organizational network analysis (2007).*

[40] Ian Robinson, Jim Webber and Emil Eifrem, *Graph Database- New opportunities for connected data, 2nd Edition (2015).*

[41] Theo Haerder and Andreas Reuter, *Principles of Transaction-Oriented Database Recovery (1983).*

[42] MYSQL, *https://www.mysql.com/*

[43] mongoDB, *https://docs.mongodb.com/manual/introduction/*

[44] CouchDB, *http://docs.couchdb.org/en/2.1.1/intro/index.html*

[45] Cassandra Database, *http://cassandra.apache.org/doc/latest/cql/index.html*

[46] HyperGraph database, *http://www.hypergraphdb.org/*

[47] NoSQL Database, *http://nosql-database.org/*

[48] Strozzi NoSQL, *https://en.wikipedia.org/wiki/Strozzi$_N$oSQL*

[49] Scalaris, *http://scalaris.zib.de/*

[50] Oracle No SQL Database, *https://www.oracle.com/database/nosql/index.html*

[51] Barry Leventhal, Head of Analysis and Modeling Teradata, *Data Mining, analysis and Modelling.*

[52] Jasmina Kandelaar, Tilburg University, Data Science: Business and Governance, *A cross-category method comparison for doing market basket analysis (2016).*

[53] Nikhil Verma, University of Venice, *Market Basket Analysis with Network of Products (2010).*

[54] Duncan J. Watts, *Networks, Dynamics, and the Small-World Phenomenon, American Journal of Sociology Volume 105, No. 2, pp. 493-527 (1999).*

[55] Linton C. Freeman, *Centrality in Social Networks Conceptual Clarification. Social Networks, Volume 1, Issue 3, 1978–1979, Pages 215-239.*

[56] Gephi - The Open Graph Viz Platform, https://gephi.org/

[57] Santo Fortunato and Marc Barthelemy, *Resolution limit in community detection (2006).*

[58] Benjamin H. Good , Yves-Alexandre de Montjoye and Aaron Clauset, *The performance of modularity maximization in practical contexts (2010).*

[59] J. W. Berry, B. Hendrickson, R. A. LaViolette, and C. A. Phillips *Tolerating the Community Detection Resolution Limit with Edge Weighting (2009), e-print, arXiv:0903.1072.*

[60] Google Charts *https://developers.google.com/chart/interactive/docs/*

[61] jQuery *http://jquery.com/download/*

Rashmi Priya

Graduate Student,
University of Kentucky,
Department of Computer Science.

Education

- Bachelor in Computer Science, West Bengal University of Technology, 2012.
- Master in Computer Science, University of Kentucky, 2018.

Professional Positions

- **System Engineer**, Tata Consultancy Services Limited 2013-2016.
- **Business Analytics Developer Intern**, Institutional Research and Advanced Analytics, University of Kentucky 2016-2017.
- **Graduate Teaching Assistant**, Department of Computer Science, University of Kentucky 2017-2018.