Predicting Unknown Adverse Drug Reactions Using an Unsupervised Node Embedding Algorithm

by

Sourav Das

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Computer Science and Engineering
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2019

© Massachusetts Institute of Technology 2019. All rights reserved.

Author ................................................................
Department of Electrical Engineering and Computer Science
September 9, 2019

Certified by............................................................
Lalana Kagal
Principal Research Scientist
Thesis Supervisor

Accepted by...........................................................
Katrina LaCurts
Chair, Master of Engineering Thesis Committee
Predicting Unknown Adverse Drug Reactions Using an
Unsupervised Node Embedding Algorithm

by

Sourav Das

Submitted to the Department of Electrical Engineering and Computer Science
on September 9, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Defined as undesirable effects of a medication that occur during or after usual clini-
cal use, Adverse Drug Reactions (ADRs) pose a major health risk and result in the
hospitalization of millions of patients each year. While pre-marketing clinical trials
evaluate the safety and efficacy of a new drug, post-marketing surveillance identifies
and monitors ADRs that were not previously identified during trials. Traditionally,
most approaches tend to focus on ADR detection in the post-marketing phase. Also
current approaches mostly use supervised machine learning, requiring significant pre-
processing of the data and feature engineering. I developed a customizable framework
based on unsupervised learning that allows users to run prediction tasks on different
types of labeled graph data. The framework first creates a knowledge graph from the
data and then uses an unsupervised algorithm to create embeddings (vector repre-
sentations) of the nodes in the knowledge graph, and finally runs the prediction task.
The framework enables an embedding to be learned for any newly added node as
long as it is connected with the other nodes, and users can create embeddings for any
pre-marketed drug as long as its related drug attributes are present in the knowledge
graph. Using DrugBank and FAERS, I created a knowledge graph of drugs and drug
attributes. To emulate drugs in the pre-marketing stage, I removed all the drug-ADR
edges in the test dataset. Then, I experimented with different parameters of the node
embedding algorithm and three different classifiers namely MLP, KNN and random
forest. The models were trained to predict 9 different ADR associations for any drug,
and our results showed that the MLP classifier was the best model with an AUROC
score of 0.79, which is comparable to existing approaches but with much greater cus-
tomizability. This approach has potential to improve how ADRs are predicted and
allow them to be detected at a far earlier stage thus improving patient safety.

Thesis Supervisor: Lalana Kagal
Title: Principal Research Scientist
Acknowledgments

I would first like to thank my thesis advisor, Lalana Kagal, for being a great supervisor and a terrific mentor. Working on this project has been an incredibly experience, and I am very thankful for Lalana’s guidance throughout the whole process.

I would also like to thank my girlfriend, Devanshi, who stuck with me during the months of writing and re-writing even when I retreated to long days with my computer.

Lastly, I would like to express my profound gratitude to my parents, Suchata and Goutam, for providing me with unfailing support and continuous encouragement throughout my years of study.

This accomplishment would not have been possible without them. Thank you.
# Contents

1 Introduction 13

2 Related Work 17
  2.1 Disproportionality Analysis (DPA) 18
  2.2 Supervised Machine Learning Approaches 19
  2.3 Unsupervised Machine Learning Approaches 20

3 Approach and Implementation 21
  3.1 Compiling Data 22
    3.1.1 DrugBank 22
    3.1.2 Curated FAERS 22
    3.1.3 OHDSI vocabulary 24
    3.1.4 RxNorm database 24
  3.2 Creating Knowledge Graph 24
    3.2.1 Extracting knowledge graph from DrugBank 26
    3.2.2 Integrating FAERS data into knowledge graph 28
  3.3 Creating Network Representation 30
    3.3.1 Implementing network representation 30
  3.4 Node Embedding with Neighborhood Aggregation Algorithm 32
    3.4.1 Model Architecture 34
  3.5 Training Classifiers 35

4 Experiments and Results 37
4.1 Knowledge Graph Summary ........................................... 37
4.2 Training and Testing .................................................... 38
4.3 Experiments with Prediction Models ................................. 39
  4.3.1 Effects of different Negative Sampling ....................... 39
  4.3.2 Effects of different Support Size .............................. 50
  4.3.3 Prediction Statistics ............................................. 60

5 Future Work & Conclusion .............................................. 63
List of Figures

1-1 Example of how a drug in DrugBank is connected with other nodes  . 15

3-1 Dataflow Diagram: Yellow boxes indicate data and red boxes indicate
functions ................................................................. 21
3-2 DrugBank XML schema ........................................... 23
3-3 General structure of Neighborhood Aggregation Algorithm [20] .... 33
3-4 The Neighborhood Aggregation Algorithm Model .................... 35

4-1 MLP Classifier Performance with Negative Sampling=10. .......... 40
4-2 KNN Classifier Performance with Negative Sampling=10 .......... 41
4-3 Random Forest Classifier Performance with Negative Sampling=10 . 42
4-4 MLP Classifier Performance with Negative Sampling=20 .......... 43
4-5 KNN Classifier Performance with Negative Sampling=20 .......... 44
4-6 Random Forest Classifier Performance with Negative Sampling=20 . 45
4-7 MLP Classifier Performance with Negative Sampling=50 .......... 46
4-8 KNN Classifier Performance with Negative Sampling=50 .......... 47
4-9 Random Forest Classifier Performance with Negative Sampling=50 . 48
4-10 MLP Classifier Performance with support sizes=(20, 15) .......... 50
4-11 KNN Classifier Performance with support sizes=(20, 15) .......... 51
4-12 Random Forest Classifier Performance with support sizes=(20, 15) . 52
4-13 MLP Classifier Performance with support sizes=(50, 25) .......... 53
4-14 KNN Classifier Performance with support sizes=(50, 25) .......... 54
4-15 Random Forest Classifier Performance with support sizes=(50, 25) . 55
4-16 MLP Classifier Performance with support sizes=(75, 50) .......... 56
4-17 KNN Classifier Performance with support sizes=(75, 50) . . . . . . . . 57
4-18 Random Forest Classifier Performance with support sizes=(75, 50) . . 58
4-19 MLP Classifier Performance with Negative Sampling=20 . . . . . . . . 61
List of Tables

4.1 Node Counts in the Graph ........................................... 37
4.2 Edge Counts in the Graph ........................................... 38
4.3 ADR label counts in training and testing sets .................... 38
4.4 Prediction Statistics of MLP classifier with support=(20, 15) and Negative sample size=20 ....................... 60
Chapter 1

Introduction

Defined by the World Health Organization as "the science and activities relating to the detection, assessment, understanding, and prevention of adverse effects or any other drug-related problems", pharmacovigilance is a key component of effective drug regulation systems. An important topic in this field is the identification of Adverse Drug Reactions (ADRs). ADRs are harmful, often unexpected interactions caused by drugs, which can occur either suddenly or gradually, during or after the administration of a drug. The impact of ADRs on the United States healthcare system is complex and is estimated to cost approximately $30.1 billion dollars annually for all the ADR-related costs such as hospitalization and prolongation of hospital stay [7]. Davies et al. found that ADRs increased a patient’s mean hospital stay from 8 to 20 days and increased the risk of mortality in patients who experienced an ADR [8]. Therefore, identifying the ADRs of a drug should be a crucial goal during the clinical trial stage. However, due to constraints such as sample size and study population, it is not always possible to detect all the ADRs of a drug in the trial stage [7]. As a result, newly marketed drugs go through post-marketing surveillance, which is not always effective due to the limited reporting of ADR occurrences, time constraints, and inconsistent reporting with certain ADRs being reported more than others [7].

Some of the largest sources of post-marketing ADR data are Spontaneous Reporting Systems (SRS) whose datasets contain the anonymized case information of ADRs [5]. In the U.S., healthcare professionals report ADRs through the Food and Drug
Administration (FDA) Adverse Event Reporting System or FAERS [5]. Most existing ADR prediction methods use FAERS datasets to find ADRs for post-marketed drugs. Disproportionality Analysis (DPA) methods are used to mine drug-ADR association rules from reporting systems [15]. Some supervised machine learning approaches use drug component information such as target proteins and pathways to improve ADR prediction [7]. ADR prediction relies on careful feature engineering, selecting what drug information to use as features, which affects the prediction quality. Since supervised prediction models require feature-specific information for all drugs, preprocessing of the data requires large amounts of effort and is necessary for the training of supervised models [15]. The majority of these models are trained using a fixed set of features and do not allow for the integration of new drug-related information.

In this project, I addressed issues posed by supervised ADR predictions such as requiring labeled data from the post-marketing phase and limiting the features that can be added to the supervised model. While exploring possible solutions, I focused on the ability to predict drug-ADR relationships for drugs in the pre-marketing stage. I created a feature generation process that allows users to leverage semantically annotated information related to drugs. Users are able to modify the model easily and add new drugs or drug component information using a simple interface. I utilized a neighborhood aggregation algorithm [20] that generates embeddings for a graph labeled with information (linked data [1]). This approach is an unsupervised inductive node embedding process that generates embeddings in a graph given the appropriate node information. The process is considered inductive due to the ability to create an embedding for a newly added node in the graph given the embeddings of its neighbors. The algorithm works for graphs whose nodes have additional information like text features or node degree statistics.

Most existing approaches that use linked data create a knowledge graph [7], or a large network of entities [15], using their semantic types, properties, and the relationships between them from different drug resources. These knowledge graphs can contain different classes of nodes such as drugs, proteins, drug indications, etc., and the edges represent the relationships between these nodes. Then, these methods select
what features to use from the knowledge graph and decide how to integrate them into the ADR predictions. My approach generates a knowledge graph from a set of linked datasets using constraints provided by the user providing greater customizability.

Figure 1-1: Example of how a drug in DrugBank is connected with other nodes

I used the DrugBank [21] database, a richly annotated resource combining comprehensive drug data with drug target and action information, along with the curated FAERS database to extract a knowledge graph containing drugs and their components. Figure 1-1 is an example of how drugs in DrugBank are connected with other drug attributes. A major difference between my approach and previous methods is that I was able to integrate the labels of nodes and edges as input information in the feature generation process. For example, Acetaminophen drug acts as a substrate (labeled Action: Substrate) for enzyme Cytochrome P450 2E1(CYP2E1). The extraction of the knowledge graph can integrate the type information of nodes, the edge relationships, and the labels describing these relationships. For instance, it would integrate Acetaminophen as a drug type node, CYP2E1 as an enzyme type node, and the edge between them is labeled as "action" edge with text data "substrate." In this process, I was able to integrate any heterogeneous drug components into the graph and utilize both the node and edge labels in the feature generation process. After extracting a labeled graph, a curated FAERS data resource was used to integrate drug indications and the ADR information into the graph. Finally, before applying
the neighborhood aggregation algorithm, the knowledge graph with labeled nodes and edges was converted into a graph with nodes and nodes’ feature vectors. A non-contextual text embedding was used to convert the texts into feature vectors, and the edges and the edge information was converted into nodes with features. Then, the neighborhood aggregation algorithm generated embeddings for each node in an unsupervised way, which can be used in any machine learning task as features. In my approach, the features generated by the node embeddings are used for predicting ADRs for new drugs in the pre-marketing stage with no ADR information. The feature generation process also addresses issues such as adding new drug information and the drug’s relationship with other drugs. Because of the inductive nature of the process, embeddings can be generated for any new node that is connected with other nodes in the graph.

My thesis has three main contributions. First, by using an unsupervised machine learning approach to generate embeddings, my approach leverages different drug features in prediction tasks such as predicting ADRs for drugs in the pre-marketing stage, drug-drug interactions, and missing edge predictions. Second, my approach utilizes linked data sources like DrugBank and FAERS and allows for the information provided in these databases to be utilized to create features in an unsupervised manner. Finally, new nodes such as drugs or drug components can be added to the graph in an inductive manner, providing the ability to add new information without redesigning the feature extraction process.

The rest of this thesis is structured as follows: Section 2 is related work, Section 3 is approach and implementation, Section 4 describes experiments and results, and Section 5 concludes this work.
Chapter 2

Related Work

There are two types of data sources that include real world evidence of different ADRs used by existing approaches: (i) Spontaneous Reporting Systems (SRS) of which FAERS is the most popular example; (ii) Observational Health Data (OHD). While SRS provide reports from pharmaceutical companies, healthcare professionals and consumers, OHD includes electronic health records, patient registries, administrative claims etc. SRS provides valuable information of different adverse drug reaction events including prescribed drugs, patient indications, adverse reactions etc. Much of the work on predicting ADRs utilizes SRSs such as FAERS for mining drug reaction signals. Clinical trials tend to be divided into four separate phases. Pre-marketing clinical trials compose the first three phases while the last phase is post-marketing monitoring. In the past, research has been conducted on predicting ADRs in the post-marketing monitoring phase. However, machine learning based ADR prediction in the pre-marketing phase remains relatively unexplored territory.

The existing approaches to predicting ADRs can be divided into roughly three categories: (i) Disproportionality Analysis (DPA) methods for mining drug-ADR association rules from SRS [5]; (ii) supervised machine learning methods utilizing different features of the drugs and ADR event reporting; (iii) unsupervised machine learning methods to utilize different feature representation techniques.
2.1 Disproportionality Analysis (DPA)

The most popular DPA methods are: Relative Reporting Ratio (RRR), Proportional Reporting Rate (PRR), and Reporting Odds Ratio (ROR) [5]. Harpaz et al.[9] used the Apriori algorithm to mine the FAERS reports and generate statistically significant association rules (e.g. \textit{Aspirin} + \textit{Warfarin} $\rightarrow$ \textit{Bleeding}). Maulik et al.[15] used a variation of Apriori algorithm in a k-partite graph with four types of nodes: (i) Drugs, (ii) Proteins, (iii) Pathways and (iv) Phenotype (ADR). The Apriori-based methods compute the Support and Confidence statistics for an association rule [15]. The support of an item set \( S(A) \) is the number of records containing A. The support of an association rule \( S(A \rightarrow B) \) is equal to \( S(A \cup B) \). The support determines how often a rule, in this case the mentioning of both A and B, is observed in the data. The confidence of a rule is determined by \( C(A \rightarrow B) = S(A \cup B) \). This is an empirical estimation of \( Pr(B|A) = \text{probability of B given that A has happened} \) [9]. The general Apriori algorithm works in two steps: (i) narrow the search space by only allowing item sets that have support count above a certain threshold; (ii) generate association rules that have confidence count above some threshold [9].

In both Harpaz et al.[9] and Maulik et al. [15], some form of a relative ratio threshold instead of confidence count was used. In Harpaz et al.[9], the method used relative reporting ratio (RRR) as proxy for confidence count: \( \text{RRR} = \frac{N \times S(A \cup B)}{S(A)S(B)} \) where \( N = \text{the total number of reports in the data.} \) This is a probabilistic estimation of \( \text{RRR} = \frac{Pr(B|A)}{Pr(B)} = \frac{Pr(A,B)}{Pr(A)Pr(B)} \) [9]. Here, \( A = \text{set of drugs like Aspirin, Warfarin, B = adverse reaction like Bleeding.} \)

Maulik et al.[15] introduced a network-based relative reporting ratio where \( A = \text{set of drugs + proteins + pathways instead of only drugs.} \) They used life science linked open datasets (LSLOD) such as DrugBank and PharmGKB and created a four partite graph including drugs, proteins, pathways and adverse reactions [15]. This approach utilizes the open linked data sources to create a system network and also uses SRS for mining association rules. One of the major drawbacks of reporting ratio approaches is a sampling variance issue [5]. To solve sampling variance issues,
Bayesian approaches like Multi-item Gamma Poisson Shrinker (MGPS) and Bayesian Confidence Propagation Neural Network (BCPNN) are used. Both methods are used in cases of smaller amounts of data and attempt to shrink the relative ratio towards the prior probability distribution [5]. Cao et al.[5] used Monte Carlo Expectation Maximization (MCEM) to generate more accurate SRS drug-ADR signals by down-weighting their associations with irrelevant drugs in case reports. The DPA methods are focused on predicting drug-ADR relationships in post-marketing analysis whereas my approach applied predicts adverse drug reactions in the pre-marketing stage.

2.2 Supervised Machine Learning Approaches

Next, I explored supervised machine learning approaches to predict ADRs. Cami et al. [4] trained a logistic regression model using structural properties of the drug-ADR network, chemical and taxonomic properties of drugs as features and trained it with SRS datasets. The best model achieved AUROC (Area Under the Receiver Operating Characteristics) of 0.87 with a sensitivity of 0.42. Bresso et al.[3] used decision trees and inductive logic programming to predict ADR profiles (rather than individual ADRs) on a database of drug, ADR and target knowledge and validated with FAERS. These methods provide ways to apply machine learning approach to predict new ADRs but are limited in validation [7]. Daniel et al.[7] addressed this issue by utilizing OHD sources like electronic health records (EHRs). The team used EHRs to mine new ADRs reducing dependency on SRS and constructed a knowledge graph with 4 types of nodes: (i) Drugs, (ii) Protein Targets, (iii) Indications, (iv) Adverse Reactions. They began by using an enrichment test for selecting only the significant features essential to predict ADRs and then produced features as “meta-drug” information from the knowledge graph and using the enrichment test [7]. The produced features were later used in different machine learning prediction models such as logistic regression (LR), decision tree (DT) and support vector machine (SVM). In logistic regression, they used Youden"s J statistic as the objective function instead of log likelihood function [7]. The result for classifying known ADRs was extremely
good (AUC 0.92). They used a cross validation scheme by removing 10% of drug-ADR edges from the knowledge graph and the method still correctly predicted 68% of the deleted edges. They also used a Natural Learning Processing (NLP) pipeline [12] and electronic health record data to validate new ADR predictions that are still not in public records. In most of the supervised machine learning approaches, the feature engineering steps are important because the performance of the prediction model is dependent on the features used, feature space, and whether the model is overfitting/underfitting the prediction tasks depending on the feature space and its dimensionality. Although some supervised approaches are in the post-marketing phase while others are in the pre-marketing phase, they are all limited by the features that can be added to the knowledge graph. However, my approach is flexible to the addition of new data in feature generation process.

2.3 Unsupervised Machine Learning Approaches

Sanjoy et al.[19] introduced a deep learning model to learn feature representations of 2D and 3D graphical structures of drugs and drug components. It simultaneously predicts ADRs and identifies molecular substructures associated with those ADRs. Remzi et al.[6] used open linked data and graph embedding methods to predict ADRs. In the feature extraction step, it explored RDF2Vec [18] approach (method similar to word2vec [16]), TransE [2] and TransD [13] approaches with the goal to capture different drug relations of the graph in vector space. Sanjoy et al. uses a graphical approach and different data from my approach. On the other hand, Remzi et al. uses graph embedding approach and predicts the drug-drug interaction which is predicting edges between drugs and not the same problem as predicting new ADRs. It also uses graph embedding approaches that are not inductive which does not offer the flexibility to add new information in the knowledge graph like my approach.
Chapter 3

Approach and Implementation

I utilized open linked health data of the DrugBank and FAERS along with user constraints, used a unsupervised node embedding approach and then ran multiple classifiers to discover Drug-ADR relationships of new drugs in the trial stage. In this chapter, I discuss my overall approach, which comprises of compiling data from various sources, generating the knowledge graph based on the user constraints, using a node embedding method and finally running a set of machine learning tasks.

Figure 3-1: Dataflow Diagram: Yellow boxes indicate data and red boxes indicate functions
3.1 Compiling Data

There are two types of data sources that include real world evidence of different ADRs [5] used by existing approaches: (i) Spontaneous Reporting Systems (SRS) of which FAERS is the most popular example; (ii) Observational Health Data (OHD). While SRSs provide reports from pharmaceutical companies, healthcare professionals and consumers, OHD includes electronic health records, patient registries, administrative claims etc. SRS provides valuable information of different adverse drug reaction events including prescribed drugs, patient indications, adverse reactions etc.

3.1.1 DrugBank

The DrugBank database is an online resource containing chemical, pharmaceutical, and pharmacological information on drugs and drug target data such as structure, sequence, and pathway[21]. The database is publicly accessible and also connected to the Life Sciences Linked Open Data (LSLOD) cloud[15]. In this project, I created a knowledge graph from DrugBank database which contains all the chemical components connected with any specific drug. The database is presented in XML format and includes information such as targets, transporters, carriers, enzymes, and pathways for each drug. Figure 3-2 is the schema of DrugBank XML database. Each drug and component is also assigned a unique identifier. The DrugBank XML is used as an input to create a knowledge graph depicting the various drugs and their relationships with targets, transporters, carriers, enzymes, and pathways.

3.1.2 Curated FAERS

Classified as a spontaneous reporting system (SRS), the FDA Adverse Event Reporting System (FAERS) serves as a database for drug safety tracking and has an influence on safety documents produced by the FDA i.e. changes in drug labels. Due to improvement in submission procedure and an increase in data standard, there are a larger number of cases in the FAERS, making it a crucial resource for regulatory study. Because the raw FAERS dataset has limitations such as duplicate reports,
incomplete information, and lack of labels using a standardized vocabulary, I used a curated and standardized version of the FAERS dataset [14].

The curated FAERS dataset organizes the case information into multiple TSV files demonstrating the relationships between drugs, indications, and outcomes (ADRs). The dataset is used to complete my graph by adding indication nodes and connecting indications with drugs. From the dataset, statistical information for drug-ADR rela-
tionships was obtained and used as the positive and negative labels for drug concepts. Furthermore, the Implementation section of this paper will discuss the integration of information.

The dataset uses OHDSI (Observational Health Data Sciences and Informatics)[14] vocabulary to map the drug concepts to RxNorm concepts and adverse reactions/outcomes to SNOMED-CT [14] (Systematized Nomenclature of Medicine – Clinical Terms) concepts. The curated FAERS dataset was generated in order to integrate indications with the drugs in the graph and to create labels for drug-ADR relationships. In addition, the standardized vocabulary helps merge this information with the drugs from DrugBank dataset.

3.1.3 OHDSI vocabulary

The drug concepts in the curated FAERS dataset are labeled with standard OHDSI (Observational Health Data Sciences and Informatics) concept identifiers, a unique identification string for each drug. I used the OHDSI vocabulary to find the RxNorm codes for the drug concepts occurring in FAERS dataset.

3.1.4 RxNorm database

The RxNorm database maps RxNorm drug concept identifiers to other drug sources’ concept identifiers. This database is then used to map the drug concept identifiers in FAERS to the drug concept identifiers of DrugBank.

3.2 Creating Knowledge Graph

As a preliminary step, I created a knowledge graph containing drugs and their properties such as targets, enzymes, proteins, pathways, transporters, carriers, and indications from publicly accessible datasets. The implementation has the ability to extract a knowledge graph consisting of a set of drugs and any of their properties from a valid XML document. The knowledge graph only extracts drugs and those properties that
are common to multiple drugs so that the system can learn embeddings for a new
drug from existing drug embeddings based on its relationship to existing drugs. I
selected all the nodes in the DrugBank database with the following properties:

- The node has an identifier. For example, Prostaglandin G/H synthase 2 (PTGS2)
  is a protein target that has an identifier: P35354

- The node can also be potentially connected with other non-drug nodes (Thus
  possibly with a new experimental drug too). For example, PTGS2 protein
  target is connected with drugs such as Acetaminophen (Tylenol), Sulindac,
  Flurbiprofen etc.

- The relationship between the node and a drug can be labeled. For example, PTGS2
  protein target is connected with Acetaminophen with an "Action" relation in
  my graph and the "Action" is: "Inhibitor"

To create the knowledge graph, I first went through the curated FAERS datasets
and used the OHDSI and RxNorm database to map the drug concept identifiers in
FAERS to DrugBank identifiers. Next, I used the mapped DrugBank identifiers to
extract all the drugs and their properties from DrugBank XML file. The output is:

- all the nodes in the graph (Drug concepts, Pathways, Enzymes, Carriers, Trans-
  porters, Targets) extracted from DrugBank database. The nodes have the fol-
  lowing properties: Concept_id = The identifier of the node; Name = Name of
  the node; Type = Type of the node; Data = list of words containing information
  about the node

- all the edges that connect the non-drug nodes with drug nodes. They have two
  properties: Type and Data. Type = The relation between the non-drug com-
  ponent and the drug. Example: "Action" is the type of the relation I chose
  between a drug and a target. Data = list of words containing information
  about the relation between the drug and the non-drug component. Example:
  Acetaminophen’s action on PTGS2 target is: "Inhibitor". Thus, the data will
  be ["Inhibitor"]
Next, I used the mapping of OHDSI drug concept to DrugBank ids to replace the DrugBank nodes with OHDSI drug concepts. This allowed me to also integrate *indications* and *ADRs* from the curated FAERS dataset into the knowledge graph.

### 3.2.1 Extracting knowledge graph from DrugBank

This section discusses my Python implementation to extract a knowledge graph from the DrugBank database. The DrugBank database was first downloaded in XML format. Using Python’s helper library, I implemented the ElementTree XML API, a function that can extract a knowledge graph from any XML structure database of drugs. I utilized the XPath syntax that enabled the use of structured data information in XML files. Prior to discussing the structure of the function, it is imperative that I understood the layout of information in DrugBank. DrugBank has assigned an ID to every drug and drug component in DrugBank. A drug can have an action listed under a drug component that it is related to.

Here is the interface description of the function:

```python
def extract_knowledge_graph(
    fpath, drug_xpath, drug_id_xpath, component_types,
    component_xpaths, id_xpaths, action_xpaths, selected_drugs):
    """
    // Comments

    // Parameters Explained
    fpath (type: string) = The path to the DrugBank XML path
    drug_xpath (type: string) = The XPath to extract every drug in the DrugBank
    drug_id_xpath (type: string) = The XPath to extract the DrugBank id
    component_types (type: list) = A list of strings with the names
        of the properties to extract
    component_xpaths (type: list) = A list of strings with the xpaths
        of the properties (in order)
    id_xpaths (type: list) = A list of strings with the xpaths
```

26
of the properties’ ids (in order)

action_xpaths (type: list) = A list of strings with the xpaths

of the action’s between the drug and the property

selected_drugs (type: list) = A list of DrugBank ids to extract in the graph.

If empty, extract all the properties

The function utilizes the ElementTree XML API to iterate through every drug and
property in the database. Every time the function iterates through a new element, it
adds the element in a hashmap of nodes with the following information:

node_id --> type, name

Here, node_id is the DrugBank ID for the component. Next, it adds the relationship
between a drug and property as follows:

(drug_id, drug_component_id) --> action_type, action

Here, it is possible to have no action if there is no action labeled between the drug
and its properties. Therefore, the function returns the extracted knowledge graph
with the following outputs:

Nodes (hashmap) = A hashmap of nodes with the keys are node_id

and the values are type and name as a list

Edges (hashmap) = A hashmap of edges with the keys are (drug_id, drug_component_id)

and the values are the action that indicates

the relation between the components

Here is an example of the function call that takes an XML data source and XPath
information of the drugs and drug attributes and produces Nodes and Edges of the
knowledge graph as specified before:
3.2.2 Integrating FAERS data into knowledge graph

The curated FAERS dataset includes the drugs and the indications using OHDSI vocabulary. To map the drug IDs from the OHDSI concept IDs to DrugBank IDs, I used the RxNorm database to convert each OHDSI concept to a RxNorm and then each RxNorm to a DrugBank ID.

Merging FAERS IDs with DrugBank IDs

I used several methods to merge the DrugBank IDs with FAERS drugs.

—**Directly Via RxNorm:** I extracted all drug IDs (OHDSI concept IDs) in the curated FAERS dataset. I used Python CSV File API to extract the drug IDs from FAERS TSV files and create a hashmap of: OHDSI_concept_id → Name. Next, I extracted OHDSI_concept_id → RxNorm_id from OHDSI TSV file (downloaded as the name conversion from OHDSI to RxNorm). RxNorm database provides mapping from RxNorm to DrugBank ID.

—**Name Merging:** The curated FAERS dataset also provides the names of the
drugs in the TSV files with the IDs. From DrugBank, I obtained DrugBank IDs of the drugs and their names. I created a hashmap with the names of drugs as the keys and the DrugBank IDs as the values. Finally, I checked which drug names can be merged from FAERS to DrugBank.

—Manual Merging: Even after using the above methods, some OHDSI drug concepts could not be merged. I decided to manually find DrugBank IDs for all the drugs that occurred at least 200 times in FAERS reporting but had not been merged via the other two processes. I had to manually process 81 drugs in manual merging. In most of these cases, the different naming and old drug concepts had not been converted to DrugBank ID via RxNorm.

Adding Drug Indications

The curated FAERS dataset contains the drug indication information for every case. Using the TSV files, I mapped the drug indications to case ID. The information of which drug was used for each case is also provided. Using the case IDs, I finally merged the drug concepts in FAERS with the appropriate drug indications.

Adding Drug-ADR labels

I used the curated FAERS dataset to connect the drugs with their reported ADRs. FAERS contains the drug-ADR pairs’ proportional reporting ratios (PRR), a statistic that is used to summarize the reporting of an ADR when a specific drug has been prescribed. Whenever PRR statistics for a drug-ADR pair is higher than 1, it is commonly suggested that the ADR is more frequently reported whenever the drug is prescribed. Therefore, whenever the PRR reporting ratio is greater than 1, I added an edge between a drug and an ADR.

After this stage, I ended up with two new types of nodes in my graph: drug indications and ADRs as properties of drugs.
3.3 Creating Network Representation

Finally, I performed one more transformation in my knowledge graph to prepare the graph for the next steps. For any labeled edge between a drug and a non-drug node, I did the following:

- I removed the edge and replaced it with a node labeled by the data of the edge and the non-drug component that it is connected to. For example, if the edge is between PTSG2 and Acetaminophen, it is labeled as Type: Action and Data: ["Inhibitor"]. Then, the new node will be: "PTSG2_inhibitor" and the type of the node will be: Action and the data: "Inhibitor".

- If data includes multiple items, then I created multiple nodes to replace that edge. For example, there is an edge between CYP2C9 enzyme and Sildenafil drug. Type: Action and data: ["substrate", "inhibitor"]. I created two nodes in this case as "CYP2C9_substrate" and "CYP2C9_inhibitor" with data [substrate] and [inhibitor] respectively.

- Any drug mode with a similar edge to a particular non-drug component will be connected to the same newly-added edge that replaces the similar edge. For example, if the (Fluconazole, CYP2C9) edge is an Action type edge with data [substrate] and the (Sildenafil, CYP2C9) edge is an Action type edge with data [substrate, inhibitor]. Then, Fluconazole and Siladenafil will both be connected to CYP2C9_substrate node and Siladenafil will also be connected to CYP2C9_inhibitor node.

The node embedding algorithm I’m using requires nodes with features, and this transformation converts a labeled edge into a node with feature information.

3.3.1 Implementing network representation

First, I converted the edges into nodes with text information. Next, I used one hot encoding [10] to convert the text information into feature vectors. Finally, I converted
my knowledge graph into a NetworkX graph format. NetworkX is a Python package that allows us to store the knowledge graph in an efficient way and provides many helpful methods for graph manipulation.

Every edge in the knowledge graph is connected to a drug type node and a non-drug type node and contains a label characterizing the relationship between two nodes. For instance, the edge is between PTSG2 and Acetaminophen is labeled as Type: Action and Data: ["Inhibitor"] . I used the non-drug ID and the action to create a new node with an ID: "Inhibitor_PTSG2". This node would have the type same as the edge type (which is "Action" in my case) and be connected to both the drug node (Acetaminophen) and the non-drug node (PTSG2).

Next, I used one-hot encoding to create feature vectors for each node. I first extracted all the type words and data words from the nodes. The one-hot encoding [10] assigned a vector to every node as follows: for every word that is in the type or the data of the node, it assigned 1 to those indices and zero to everything else.

Finally, I created a NetworkX undirected graph by adding all the nodes and edges. I then stored the feature vector created by the one-hot encoding as a node attribute of the NetworkX graph and created class labels for every drug components and assigned class labels as attributes of the graph. I randomly selected 20% of drug nodes as my test set. To emulate drugs in pre-marketing stage, I removed all the edges between the drugs and the ADRs in my test set. Therefore, the test set will have edges only with other drug components. I added an additional attribute for every node indicating if the node is in the test set or not.

Thus, the final graph is a NetworkX graph with nodes having three attributes: Feature_vector, Class_label, and Test_label. As the variable name suggests, Feature_vector represents the feature vector associated with the node. The Class_label refers to the drug nodes to be used in my prediction model to measure ADR prediction performance, and the Test_label refers to my neighborhood aggregation model and prediction model to distinguish which nodes to use for training and which ones for testing.
3.4 Node Embedding with Neighborhood Aggregation Algorithm

A node embedding for a graph is a representation of the nodes in the graph that can be utilized in graph-related machine learning problems such as edge finding or classification. The goal is to map the nodes into an n-dimensional vector space so that similar nodes/structures will have similar representations. Several node embedding architectures that are popular in representation learning include matrix factorization encoder-decoder, random walk encoder-decoder, neighborhood autoencoder, neighborhood aggregation and convolutional autoencoder [20]. The general encoder-decoder approaches such as matrix factorization or random walk have few drawbacks including: i) parameters are not shared between the nodes; ii) does not leverage node attributes; and iii) cannot generate embeddings for nodes outside training stage [20]. Although neighborhood autoencoders try to solve the first drawback by taking into account of the graph structure, they still have some serious limitations like large input dimension, costly operations, and sometimes the solution is intractable for large graphs [20]. The neighborhood aggregation and convolutional autoencoder approach attempts to solve these issues by taking the node features into account and learning an embedding function that can output a node’s embedding through the related features and the neighbors [20]. I will be using this approach with my networkx representation to learn embeddings not only for the nodes in the datasets but also for new nodes that may be added in future. Also, this approach is scalable as it requires only to search for (k-distance) neighbors of a node to find representation and allows us to find representations for newly added nodes that have not been in the graph in training stage [20].

Once I created the feature vector for every node in the knowledge graph, I used the neighborhood aggregation algorithm to find node embedding for every node in the graph. I utilized the node embedding as feature vectors to train my machine learning model. The standard form of neighborhood aggregation is shown in Figure 3-3.

The neighborhood aggregation methods built up representation for a node in a re-
cursive manner as it initialized the embeddings for all node by input features. At each step of iteration, it aggregated the embeddings of their neighbors by an aggregation function that takes a set of vectors as input. After the aggregation, the embedding of each node is updated using the weighted aggregated neighborhood vectors and the previous embedding. The trainable parts of the algorithm are the aggregation function and the weight vectors. Different versions of neighborhood algorithms can be used including graph convolutional networks (GCN) [20], column networks [20], and GraphSAGE [22]. All these implementations of this algorithm primarily differ in aggregation (line 4) and vector combinations (line 5) [20]. I used the general framework of neighborhood algorithms of GCN and GraphSage. I used mean aggregator as part of the aggregation method and concatenated the neighbor embeddings during each iteration as part of the vector combination approach. A mean aggregator uses the element-wise mean on a set of vectors.

This method is efficient because the parameters shared for each node embedding are significantly smaller than the dimension of the complete graph [20]. In addition, this method can learn embeddings for new nodes in the graph that were not in the training phase. Prior to training, I removed 20% of the drug nodes and learn the embeddings after training my weight matrices. Then, I evaluated the performance of my embedding on the test set of drug nodes.

Figure 3-3: General structure of Neighborhood Aggregation Algorithm [20]
3.4.1 Model Architecture

This section explains the architecture of my neighborhood aggregation algorithm. Every graph convolution layer of the neighborhood aggregation is composed of the following parts:

**Support Matrix** = For every layer, I chose a fixed number of neighboring nodes to consider called the support_size. If a node happened to have more neighbors than the support_size, I chose a random subset of them. If a node had less neighbors than support_size, I chose the nodes after doing a random walk from the node. Essentially, choosing the support_size helped to deal with random number of neighbors for every node. I referred to these neighboring nodes as support nodes and created a support matrix by listing the support nodes for each node in the graph.

**Aggregation Layer** = The aggregation layer uses the Support Matrix and the embedding output of previous layer to apply my aggregation function.

**Combination Layer** = The combination layer applies the combination function to the output of aggregation layer and the embedding of the node provided by the previous layer.

**Dense Layer** = The dense layer is a fully connected neural network layer, which acts as a non-linear transformation of the embeddings using weight matrix.

I used two graph convolutional layers in my embedding generation model. In the first graph convolutional layer, I used element-wise mean aggregation and concatenation for my combination method while the dense layers used ReLu activation. The output of the second layer is then used as embeddings. Figure 3-4 shows the architecture of my embedding model.

Here, the trainable parameters are the weights of the two dense layers. After I obtained the embeddings, I used a loss function that is used in back-propagation to train the weights of dense layer. In this case, I used a loss function described in GraphSage [22] that optimizes for distance between the nodes. The goal of the loss function is to keep the embeddings of close neighbors similar while the embeddings of the distant nodes have large difference in embedding. A distance based loss function
The Neighborhood Aggregation Algorithm Model will be as follows:

\[
J_G(z_u) = \sum_{(u,v) \in E} (-\text{Dist}(z_u, z_v)) - Q \cdot E_{(u,v) \not\in E}[-\text{Dist}(z_u, z_w)]
\]

\( G = \text{Graph}, E = \text{Edges}, z_u, z_v, z_w = \text{final embeddings of u, v, w.} \) This loss function awards the algorithm if the distance between closeby nodes are small. However, to prevent the algorithm to create similar embeddings for all nodes, negative sampling is utilized. \( Q \) determines the negative sample size. The negative samples are chosen randomly from the nodes that are far away from the current node.

### 3.5 Training Classifiers

After obtaining the embeddings from the node embedding algorithm, I used the embeddings of the drugs as features in my classifiers. While semi-supervised embedding learning was possible during the neighborhood aggregation phase, it was preferable for the embeddings to be able to be used for any prediction task since using labels to generate embeddings would make embeddings favorable for a particular prediction tasks only. I experimented with 3 different classifiers: K-Neighborhood classifier,
Random Forest classifier, and Multi-Layer Perceptron classifier.

i) K-Neighborhood classifier: Essentially, the principle underlying the nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point and use them to predict the label. The number of samples can be a user-defined constant as in the case of k-nearest neighbor learning.

ii) Random Forest classifier: Creates multiple decision trees from a sample drawn with replacement (bootstrap sample) from the training set and averages the predictions of the decision trees to output the final result. The purpose of this is to decrease the variance of the forest estimator.

iii) Multi-Layer Perceptron classifier: A class of feedforward neural network containing at least three layers of nodes: input layer, hidden layer, and output layer. I can think of MLP as a type of logistic regression classifiers in which the input is first changed using a learnt non-linear transformation. This non-linear transformation is learnt through training the hidden layers.

Note: for some classification problems, I had to deal with very imbalanced class labels. In those cases, I used random oversampling techniques to take make the number of class labels more balanced. Also, I utilized sklearn [17] optimizer, GridSearch, which optimized the parameters for my models. It essentially run all the combinations of given parameters and chose the best scoring model.
Chapter 4

Experiments and Results

In this chapter, I discuss the experiments I carried out on my framework. I describe my experimentation with different prediction models and measure their performance. I also experimented with two important parameters of the neighborhood algorithm: Negative Sampling Size and Support Size in each layers and discuss those results as well.

4.1 Knowledge Graph Summary

Tables 4.1 and 4.2 present the summary information of the knowledge graph I created.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADRs</td>
<td>17609</td>
</tr>
<tr>
<td>Actions</td>
<td>4280</td>
</tr>
<tr>
<td>Indications</td>
<td>3091</td>
</tr>
<tr>
<td>Drugs</td>
<td>2736</td>
</tr>
<tr>
<td>Targets</td>
<td>2281</td>
</tr>
<tr>
<td>Pathways</td>
<td>1146</td>
</tr>
<tr>
<td>Category</td>
<td>1146</td>
</tr>
<tr>
<td>Enzymes</td>
<td>265</td>
</tr>
<tr>
<td>Transporters</td>
<td>155</td>
</tr>
<tr>
<td>Carriers</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 4.1: Node Counts in the Graph
### Edges Counts

<table>
<thead>
<tr>
<th>Edges</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>(carriers, actions)</td>
<td>61</td>
</tr>
<tr>
<td>(drug_concept, ADR)</td>
<td>1918637</td>
</tr>
<tr>
<td>(drug_concept, actions)</td>
<td>30886</td>
</tr>
<tr>
<td>(drug_concept, category)</td>
<td>6125</td>
</tr>
<tr>
<td>(drug_concept, indication)</td>
<td>63651</td>
</tr>
<tr>
<td>(enzymes, actions)</td>
<td>549</td>
</tr>
<tr>
<td>(pathways, category)</td>
<td>1146</td>
</tr>
<tr>
<td>(targets, actions)</td>
<td>3369</td>
</tr>
<tr>
<td>(transporters, actions)</td>
<td>301</td>
</tr>
</tbody>
</table>

Table 4.2: Edge Counts in the Graph

### 4.2 Training and Testing

I split the drugs in the knowledge graph into training and testing datasets. 2237 drugs (80%) were used in the training phase and 499 drugs (20%) were used in the testing dataset. To emulate drugs in pre-marketing stage, I removed all the edges between drugs in the test dataset and ADRs. I selected the ADRs according to the following paper [7] and nine ADRs to measure the performance of my model. One major difference between my method and the method described in Bean et al. is that I was able to obtain a really large training and testing dataset. Table 4.3 shows the list and count of the labeled ADRs and drugs.

<table>
<thead>
<tr>
<th>ADRs</th>
<th>Training counts</th>
<th>Test counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akathisia</td>
<td>331</td>
<td>74</td>
</tr>
<tr>
<td>Alopecia</td>
<td>807</td>
<td>180</td>
</tr>
<tr>
<td>Amenorrhoea</td>
<td>397</td>
<td>93</td>
</tr>
<tr>
<td>Galactorrhoea</td>
<td>209</td>
<td>48</td>
</tr>
<tr>
<td>Hyperprolactinaemia</td>
<td>289</td>
<td>59</td>
</tr>
<tr>
<td>Neuroleptic Malignant Syn-</td>
<td>318</td>
<td>69</td>
</tr>
<tr>
<td>drome</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pericarditis</td>
<td>692</td>
<td>163</td>
</tr>
<tr>
<td>Pulmonary Embolism</td>
<td>1054</td>
<td>241</td>
</tr>
<tr>
<td>Stevens-Johnson Syndrome</td>
<td>550</td>
<td>120</td>
</tr>
</tbody>
</table>

Table 4.3: ADR label counts in training and testing sets
4.3 Experiments with Prediction Models

In this section, I discuss my experimentation with different prediction models and measure their performance. I experimented with two important parameters of my neighborhood algorithm: Negative Sampling Size and Support Size in each layers. Note: I use two graph convolution layers in my neighborhood models. Therefore, I had to choose a support size for Aggregation layer 1, a support size for Aggregation layer 2, and Negative Sampling Size for measuring output loss and tuning my weight matrices. In each of these experiments, I produced a Receiver operating characteristic curve (ROC curve) and a Precision-Recall curve (PR curve) for each classifier that I experimented with. The ROC Curve summarizes the trade-off between the true positive rate and false positive rate. On the other hand, the PR Curve summarize the trade-off between the true positive rate and the positive predictive value for a predictive model. As mentioned before, I used the oversampling approach in cases where the class labels were imbalanced. I also used SKlearn GridSearch to obtain optimal parameters for my models.

4.3.1 Effects of different Negative Sampling

Negative sampling is an essential parameter in my loss function and determines how much I need to take into account distant nodes and their embeddings. A large negative sampling means that the loss function will be prioritized over keeping the embeddings of two distant nodes different. On the other hand, a small negative sampling means that the model will try to make the embeddings of nearby nodes as similar as possible. I fixed the support size of the Aggregation layer 1 as 20 and Aggregation layer 2 as 15. Next, I experimented with negative sampling size: 10, 20 and 50. For every model, I calculated both AUROC scores and PR scores for all nine ADRs.
Negative Sampling Size: 10

Figure 4-1: MLP Classifier Performance with Negative Sampling=10.
Figure 4-2: KNN Classifier Performance with Negative Sampling=10
Figure 4-3: Random Forest Classifier Performance with Negative Sampling=10
Negative Sampling Size: 20

Figure 4-4: MLP Classifier Performance with Negative Sampling=20
Figure 4-5: KNN Classifier Performance with Negative Sampling=20
Figure 4-6: Random Forest Classifier Performance with Negative Sampling=20
Negative Sampling Size: 50

Figure 4-7: MLP Classifier Performance with Negative Sampling=50
Figure 4-8: KNN Classifier Performance with Negative Sampling=50
Figure 4-9: Random Forest Classifier Performance with Negative Sampling=50
Summary of the Negative Sampling Effect: From the figures, it is clear that ROC and PR curves that MLP and Random Forest are better estimators than the KNN classifier. With a small negative sampling number like 10, the precision for imbalanced classes drops as compared to 20. If the number is too large, the precision also drops due to the model being affected by too much negative sampling. The MLP classifier was slightly better than the Random Forest estimator in most cases.
4.3.2 Effects of different Support Size

Next, I experimented with different support sizes and fixed the negative sampling to 20. I used \((20, 15)\), \((50, 25)\) and \((75, 50)\) for (layer 1 support, layer 2 support).

Layer 1 support size=20, Layer 2 support size= 15

Figure 4-10: MLP Classifier Performance with support sizes=(20, 15)
Figure 4-11: KNN Classifier Performance with support sizes=(20, 15)
Figure 4-12: Random Forest Classifier Performance with support sizes=(20, 15)
Layer 1 support size=50, Layer 2 support size= 25

Figure 4-13: MLP Classifier Performance with support sizes=(50, 25)
Figure 4-14: KNN Classifier Performance with support sizes=(50, 25)
Figure 4-15: Random Forest Classifier Performance with support sizes=(50, 25)
Layer 1 support size=75, Layer 2 support size=50

Figure 4-16: MLP Classifier Performance with support sizes=(75, 50)
Figure 4-17: KNN Classifier Performance with support sizes=(75, 50)
Figure 4-18: Random Forest Classifier Performance with support sizes=(75, 50)
Summary of the Support Size Effect: As the support sizes of the layers increases, the prediction models did poorly on the imbalanced datasets. Again, the KNN classifier did worse than the other two classifiers. The precision for MLP classifier was slightly better than Random Forest estimator in the cases of imbalanced datasets. Overall, increasing the support means that I am merging more information with the nodes and the embedding most likely became somewhat more homogeneous since my graph contains a more dense population of edges rather than nodes. Therefore, the best fit parameters here are: (20, 15).
4.3.3 Prediction Statistics

This section discusses the statistics for one of the most optimal prediction models: MLP classifier with (20, 15) as the support size and 20 as the negative sample size.

<table>
<thead>
<tr>
<th>ADRs</th>
<th>Correctly Counts</th>
<th>Predicted Counts</th>
<th>Test counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akathisia</td>
<td>44</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>Alopecia</td>
<td>123</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>Amenorrhoea</td>
<td>64</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>Galactorrhoea</td>
<td>28</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>Hyperprolactinaemia</td>
<td>41</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>Neuroleptic Malignant Syndrome</td>
<td>43</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>Pericarditis</td>
<td>126</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>Pulmonary Embolism</td>
<td>190</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>Stevens-Johnson Syndrome</td>
<td>64</td>
<td>120</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.4: Prediction Statistics of MLP classifier with support=(20, 15) and Negative sample size=20
Maulik et al.’s [15] mechanism-based method to predict ADR had 0.7 to 0.8 AU-ROC scores in different validation sets. Daniel et al.’s [7] supervised methods predicted 68% of the missing drug-ADR relations in their graph. Most other ADR prediction methods have made prediction on drugs with post marketing information. The few approaches that explore methods for predicting ADR for new drugs, Maulik’s mechanism-based method’s results and Daniel’s supervised method’s results
are closely similar to what I am trying to achieve through my prediction model. From the ROC curve, one can see that almost all ADRs except for one had a >0.7 AU-ROC score. The average AUROC score was 0.79. In my model, I correctly predict around 69% of missing drug-ADR relations. This indicates that my prediction model contains better predictive power than a random prediction model in cases where there is good balance of positive and negative data points. Additionally, this prediction model is also comparable with other approaches like Maulik’s mechanism-based method and Daniel’s supervised methods on a knowledge graph but with far greater customizability. There were only three ADR cases where there was a hash imbalance of datapoints: Neuroleptic, Malignant Syndrome, and Hyperprolactinaemia for which a specific model would have been more appropriate.
Chapter 5

Future Work & Conclusion

Machine learning has the potential to become an invaluable tool for the practice of pharmacovigilance, which deals with the detection, assessment, understanding, and prevention of adverse drug reactions (ADRs) or any other drug-related problems. Accounting for 4.2-30% of hospital admissions in the United States and Canada, ADRs have a complex set of costs in clinical practice such as drug-related hospital admission [11], prolongation of hospital stay, and emergency department visits. They have been characterized as a source of economic burden on patients, their caregivers, and the healthcare facility that treats them. My study focused on predicting ADRs for drugs in the pre-marketing phase using publicly accessible labeled datasets including DrugBank, a resource containing chemical, pharmaceutical, and pharmacological information on drugs as well as FAERS, a spontaneous reporting system (SRS) that serves as a database for drug safety tracking.

In this project, I utilized the DrugBank and FAERS labeled datasets to create a knowledge graph. My framework allows users to specify mappings and builds the knowledge graph based on that mapping. I created a knowledge graph with over 32,000 different nodes and approximately 2 million edges, with at least 60,000 being non drug-ADR edges. I also integrated both node and edge text data in the feature generation process. Then, I used an unsupervised node embedding algorithm to perform ADR predictions for drugs with no past ADR information. The features produced by the algorithm were used to build classifiers for 9 different ADRs. I
tested my models on drugs (whose edges linking ADRs had been removed) to emulate drugs in the pre-trial stage. I trained my model with 2237 drugs and used 499 drugs in my test dataset. Unlike many other studies in this area, I tested my training model with a large number of data points and managed to achieve 0.79 AUROC score and predicted 69% of the missing drug-ADR edges correctly. However, for three out of nine ADRs, my model failed to make a statistically significant prediction due to the small number of data points associated with those ADRs. Since I did not have an adequate number of data points for those particular ADRs (meaning they were relatively uncommon), my model’s prediction abilities were less accurate. As a result, I realized that a generalized classifier might be more appropriate for ADRs with a small number of datapoints.

I developed a customizable framework that has the ability to extract any nodes and edges from a graph database and create a knowledge graph. Unlike most work done in the past, my approach focuses on making ADR predictions based on pre-marketing drug data. In addition, this approach provides the user with flexibility to run a range of prediction tasks and applies a modified neighborhood aggregation algorithm to take advantage of the labels on edges while creating embeddings. The promising results of this study make this an area that we should continue to explore.

In this study, I utilized drug information available in DrugBank and FAERS. Future work could involve the integration of molecular features (molecular weight, rotational bond count, etc.) of drugs into the knowledge graph. We can gain additional drug features by integrating more graph databases with drug information. For example, Life Sciences Linked Open Data (LSLOD) is a great source for linked open databases related to drugs and drug attributes. To take it a step further, these features could be incorporated in the knowledge graph as edge weights. Since including chemical features would reflect a change in the input data, it could potentially also increase the performance model’s accuracy.

My framework can be universally applicable and be used to make predictions in any field by making the feature generation process more generalized. Other avenues for future work include experimenting with various loss functions, layers, and aggre-
gation approaches, which would lead to different results. In addition, this approach could be extended to work with multiple hospitals that train their models in a decentralized manner minimizing the sharing of patient data. Looking ahead, this system has the potential to transform the healthcare system by dynamically generating more accurate ADR predictions, decreasing the burden on health professionals and regulatory agencies, and tackling the under-reporting challenges of post-marketing surveillance in the status quo.
Bibliography


[10] David Harris and Sarah Harris. *Digital design and computer architecture (2nd ed.)*.


