Investigating the Effects of Data Alignment, Augmentation, and Dataset Partitioning on Link Prediction Performance

P6

Bachelor project, 6. semester
CS-23-DV-6-01

Aalborg University
The technical faculty of Computer Science
Selma Lagerløfts Vej 300
DK-9220 Aalborg Øst

AALBORG UNIVERSITY

STUDENT REPORT

Faculty of Computer Science
Data science
Selma Lagerløfs Vej 300
9220 Aalborg Øst
http://cs.aau.dk

Theme:

Data analysis within the biological domain

Project:

P6 - Drug repurposing

Period:

February 2023 - May 2023

Project Group:

CS-23-DV-6-01

Participants:

Storm King-Isager
Maja Pipaluk Ploumann

Supervisor:

Matteo Lissandrini

Page Number: 26 + front page
Project finished: February 13, 2024

Abstract:

This project focuses on studying different link prediction models on graphs and investigating the impact of data changes, such as adding more data, on the performance of these models. The project also explores data alignment techniques to incorporate additional data into the graph and examines the potential benefits of data augmentation in improving link prediction capabilities. Additionally, the project addresses the challenge of dealing with pre-defined partitions of training, validation, and testing sets in graphs, which may suffer from data leakage, and proposes a method to create a new transductive partition without data leakage. The effects of these different partitions on the performance of link prediction algorithms are also analyzed. Although the project did not utilize hyperparameter optimization, it successfully quantified the effects of data leakage and observed the impact of merged datasets. The results highlight the potential for improved results through hyperparameter optimization and demonstrate significant improvements in certain scenarios, such as the ComplEx model with the merged split and new edges. The findings emphasize the importance of selecting the right dataset and split for link prediction tasks while acknowledging the potential for further enhancements through hyperparameter optimization.

Preface

This project was written by the group CS-23-DV-6-01, a group of two students, as the bachelor project at the sixth semester of the bachelor in Data Science. Matteo Lissandrini was supervisor on this project.

Aalborg University, February 13, 2024

Maja Pipaluk Ploumann
<mploum20@student.aau.dk>

Storm King-Isager
<skingi20@student.aau.dk>

1 Summary

We know that the task of drug development can be a long and expensive process. To decrease the cost and time of drug development, drug repurposing can be used. Drug repurposing can be done by using link prediction. Link prediction is used on graphs, in our case we have three different graph datasets. PharmKG8k, CTD_DDA and NDFRT_DDA, where CTD_DDA and NDFRT_DDA are from the same GitHub called BioNev and therefore on the same format. The BioNev dataset does not contain relation types and the datasets can not have any data leakage since any entity-to-entity connection only exists once. The PharmKG8k dataset has relation types and is much bigger than the BioNev datasets, it has around 400,000 edges whereas the CTD_DDA dataset has around 100,000 and the NDFRT_DDA dataset has around 50,000. The BioNev datasets has what is called a UMLS_CUI which is an ID, but the PharmKG8k dataset does not. The intention of using all the datasets is to at some point merge them, therefore a common ID is needed. We use the UMLS dataset, which contained the original names of the entities in PharmKG8k and the UMLS_CUI, so we could give the PharmKG8k data a UMLS_CUI.

Also using the UMLS_CUI it was possible to make an API search on the UMLS_CUI website and get the semantic types of each drug or disease. Here we found out that NDFRT_DDA had a semantic type that CTD_DDA and PharmKG8k did not. This semantic type constituted 64.10% of all of NDFRT_DDA's entities. These semantic types were really specific including e.g. specific dosage, therefore we again use the API search to link the specific drug, to a more general form, and in the end, we could add 7,380 new edges to NDFRT_DDA. This also made the connectivity higher between NDFRT_DDA and CTD_DDA, PharmKG8k.

By analysis of the original splits of CTD_DDA, NDFRT_DDA and PharmKG8k, we found out the CTD_DDA, NDFRT_DDA could not have data leakage since there are no entity-to-entity combinations there is there more than one time. The splits, however, were inductive with around 900 unseen entities in the test.

In the split analysis of PharmKG8k, we found out that the split, had a lot of data leakage, between all the sets. Internally in each set and between each set, there were also a lot of 100% duplicates, where the edges are equal. Due to a lot of data leakage, there were not many unseen entities, under 12 for both the test set and the validation set. We did not expect that PharmKG8k where not transductive since it is found in the Pykeen library under transductive splits.

This gave us the opportunity to investigate what happens when you try to use an inductive split without data leakage in Pykeen, which moves the edges with unseen entities from the test set and validation set, thereby downsampling the test and validation set. By chance when aligning the PharmKG8k data to a UMLS_CUI we got a transductive split. Also in this process, we found out that, PharmKG8k includes synonyms, so some PharmKG8k entities were linked to the same UMLS_CUI.

Now we had a transductive split, with data leakage, and we would now like to make several different splits so that these could be compared with the previous one. For PharmKG8k we made another transductive split without data leakage and an inductive split without data leakage. For NDFRT_DDA we made two different transductive splits without data leakage, one with the original NDFRT_DDA and another with the new edges added to it. For CTD_DDA we only made a transductive split. In addition, we made two merged splits, both were transductive and without data leakage, one with all the datasets and one where we also added the new edges from NDFRT_DDA.

We tried to reproduce the results from the PharmKG8k paper, but that was not possible. Therefore we used the hyperparameter from the templates on the Pykeen GitHub. The models we choose were ComplEx, ConvKB and TransE. By using the different splits in the models, we could clearly see, that the splits which had data leakage had high accuracy, which where expected. Also NDFRT_DDA with new edges, most of the time had a higher accuracy than without the new edges, this was also true for the merged data having the NDFRT_DDA with new edges. However, an overall best split between all the models, where not possible. This is most likely due to the fact that we did not do hyperparameter optimization, due to time constraints. We made a proof-of-concept hyperparameter optimization for NDFRT_DDA, and the result of this was a much better model, that outperformed all the others by far.

Contents

2 Introduction

The task of developing a new drug is a time-consuming and expensive process. Even though the money and time have been spent, there is no guarantee that the drug will be approved [1]. The drug typically has to go through a series of steps, including drug discovery, preclinical testing, clinical trials, and regulatory approval [2].

Given the time-consuming and expensive process, it makes sense to try and find different alternatives to this process. The opportunity of using already existing drugs for other diseases can be both time and money-saving. This concept is called repurposing and has become more feasible in the era of big data, and therefore a possible alternative [3]. The right data together with the right model can be used to do wonders for finding promising repurposing opportunities, by applying sophisticated machine learning algorithms to identify new connections and relationships between drugs and diseases.

These machine learning algorithms usually use data in the form of graphs. A graph is a collection of nodes, with edges between some of them. If you have the different drugs and diseases as nodes, and then connections between these, it makes it possible to find new links between nodes, by using link prediction, to potentially find already existing drugs that can treat already existing diseases [4].

The data which these machine learning algorithms are given have a great impact on the results of the prediction, this data is usually split up into a training, validation and testing set, to check the model's performance on new data and to flag for problems like overfitting or selection bias [5]. With that in mind, the split becomes really important, for instance, overlapping edges between the sets, can cause results which are not representative, caused by training and testing on the same edges. The importance of a transductive split without data leakage will be elaborated in Section 4.3. A transductive split is used when wanting to perform transductive link prediction. Transductive link prediction only aims to predict missing links between already existing nodes, i.e. it can't predict links between nodes which it has not seen in the training set. Therefore the properties of a transductive split are as follows; the validation and the test set never have any entities and relation types that were not present in the train set.

Since it takes a lot of time and resources to gather new data and make splits, people often use already gathered data and premade splits. But what if the premade split has data leakage or it is an inductive split, can this affect the results of, for instance, testing a new model or someone trying to repurpose a drug? It should be noted that when we refer to inductive splits, we still plan to do transductive link prediction on these using Pykeen [6], however, Pykeen removes all edges from the validation and test set, which causes the split to be inductive, and by so makes the split transductive instead. So the issue we want to investigate is the effect of downsampling. So when we refer to inductive splits, it is actually the effect of downsampling that is getting investigated.

This is the case for the PharmKG8k data, which is a part of the Pykeen package for Python. PharmKG8k is a comprehensive medical heterogeneous knowledge graph, where the nodes consist mostly of genes, but also different drugs and diseases among other things, in addition, it has 28 different relation types. Even though the split is available on the Pykeen library under the category of transductive link prediction, the split is not transductive and for instance, has 28,086 overlapping edges between train and test.

In this project, we plan to compare the results from the original PharmKG8K split with data leakage, and the results from doing transductive link predictions on different transductive splits without data leakage and different inductive splits. The goal of comparing the different splits is to investigate the split's impact on the results.

To get a more comprehensive investigation, will we in addition use the datasets CTD_DDA and NDFRT_DDA available on the BioNev GitHub [7]. The datasets are like the PharmKG8k heterogeneous knowledge graphs. Both CTD_DDA and NDFRT_DDA have drug-disease interactions, and they are simple graphs without relation types and are also used for link prediction in the biomedical domain. It is also wanted to investigate whether joining these datasets with PharmKG8k will yield better results, which by intuition should be true, since in machine learning more data usually means better models [8].

Lastly, an improvement to NDFRT_DDA will be proposed, which should improve the connectivity between NDFRT_DDA and the other datasets. This is because that 64% of NDFRT_DDA nodes are of a semantic type, that none of the other datasets has. The reason for this is that NDFRT_DDA is based on drug bank [9], which instead of having a broad category for each drug, has the specific medicine, sometimes both including the route of administration and the specific dosage. This improvement will be tested, and it will be seen whether improving the connectivity between these, will produce better results.

The code and the data used to make the splits, and the models are available on our GitHub.

3 Problem analysis

A graph consists of sets of nodes forming edges, these nodes are often denoted as V for vertices. The edges, often denoted as E signify a link between two nodes. By so a graph can be defined as the following: $G = (V, E)$, where V are the nodes and E are the edges. If

$E \subseteq |V| \times |V|$ represents the true links between entities in the network, then we are given a set of observed links which we call E_O, then $E_O \subset E$ and the aim of link prediction is then to identify the unobserved links [10].

What we plan to do is transductive link prediction [11]. If you have the dataset partition where D_{tr} is the training set, D_{ts} is the testing set and D_{va} is the validation set. Then the dataset being transductive can be expressed as such:

$$\forall \; nodes \; V \in D_{ts} \cup D_{va} \subseteq \forall \; nodes \; V \in D_{tr}$$

The ability to be able to predict a link between two entities can be used in many domains, such as drug repurposing or recommendation systems. By using link prediction, you can potentially reduce the time and cost of repurposing a drug, but at the same time can bad link prediction with low accuracy lead to missed opportunities or dead ends in the context of drug repurposing.

When wanting to make link prediction, it is necessary to have data. In our case, we use datasets with the following names: PharmKG8k [12], CTD_DDA [7] and NDFRT_DDA [7]. CTD_DDA and NDFRT_DDA are in the same format with common IDs and have no relation labels. PharmKG8k however has a different ID and relation labels. We would like to be able to merge these datasets, to investigate how the size of the data used for link prediction, can affect the results. Using any data to make link predictions, will often include splitting the data into a train, validation and test set. But how do we integrate the data and what can lead to results which are not representative?

The one thing which the datasets have in common is that they all are in a graph format. The difference between them is that CTD_DDA and NDFRT_DDA have no relation types, whereas PharmKG8k has relation types. This leads us to the fact that there exist different kinds of graphs. These are mainly distinguished by if the nodes can have different types, and if the edges can. In a homogeneous graph, every node and edge is of the same type. In a heterogeneous graph, the nodes and/or the edges have different types. One example of a homogeneous graph could be a social network, such as Facebook. Here each node is a person, and each edge is a friendship between two persons.

An example of a heterogeneous graph could be a drug-disease interaction knowledge graph. Here a node can either have the type "disease" or "drug". If this knowledge graph also contains drug-drug interactions, the edges can also have different types, such as two drugs that interact could have the relation type "interact", or that one drug is and sub-drug of another, while an edge between a drug and disease might have the relation type "treats", or "enhance", if a drug should not be taking while having a specific disease. With that in mind, how do we merge the datasets when we have two different formats? Hereby including, graphs with and without relation types, graphs without a validation set and one with, and graphs with different ID's for their entities.

3.1 Problem statement

The objective of this project is to study different link prediction models on graphs. Furthermore, we want to investigate whether a change in the data, such as having more data has an effect on the performance of different link prediction algorithms. To investigate what effect adding more data has, it is necessary to do data alignment in order to add more data, so another problem is figuring out how this can be done. Since some of the added data is not well connected to the rest of the graph, it is wanted to investigate how data augmentation can be done, and if it will have an effect to further improve the link prediction capabilities of the models on the graph. Lastly, it is wanted to investigate what can be done when a graph has a premade partitioning of the training, validation and testing sets, which both are inductive and with data leakage, and how to make a new partition which is transductive without data leakage. In addition, it will be explored what effects these different partitions of the training, validation and testing sets have on the performance of the link prediction algorithms.

4 Theoretical framework: Neural networks, graph embedding, and dataset partitioning

4.1 Neural networks

A neural network is inspired by the human brain [13]. One way to use a neural network is in the context of predicting whether two nodes in a knowledge graph could have a link/edge between them. This is called link prediction. In the context of finding a potential new purpose for a drug, link prediction and thereby neural networks, can be really useful.

The problem of link prediction has been treated in many ways, one way is with graph representation learning approaches and in particular with deep learning approaches for graph representation learning [10]. To explain this we will need to understand neural networks, so that will be explained in this section.

Introduction to Neural networks

All neural networks, also called artificial neural networks (ANN) have an input layer and an output layer, and most also have a hidden layer or layers. In general can, all layers can have multiple nodes/neurons. Each neuron receives inputs. The input layer receives input from the data, the hidden layers receive input from the input layer or other hidden layers, and the output layer receives input from the last hidden layer. Each neuron forms its input to output through its own function f [14].

There are different types of neural networks, and the type of the neural network depends on the way each neuron is connected. In figure 1, 3 different types of neural networks can be seen. Where 1 is the Perceptron-type neural network, where we only have an input layer and an output layer. It works by receiving input and then the weighted input is computed for every node. Then, to facilitate classification, an activation function is applied, typically a sigmoid function.

Number 2 is the Feed Forward neural network type. The types consist of perceptrons arranged in layers without forming cycles. Each perceptron in one layer is fully connected to every node in the next layer, but there are no connections between nodes within the same layer. There are no Back-loops, and therefore the backpropagation algorithm is often used to update weight values and minimize prediction errors.

Number 3 is the deep feed-forward neural network type. The difference between number 2 and number 2, is that a deep feed-forward network employs multiple hidden layers. Since overfitting can be a problem when using just one hidden layer, adding more hidden layers can potentially address this issue and enhance generalization [15].

To understand and compare the predictions based on a neural network, it is necessary to understand the hyperparameters of a neural network. Since these hyperparameters can differ from neural network to neural network, because of that, the same data can give different predictions, depending on the hyperparameters.

In the following, we will try to break down the different hyperparameters. Which will provide the necessary knowledge to understand and compare neural networks and thus predictions based on neural networks. The hyper-parameter that will be described is the learning rate, the batch size, the epochs and the embedding dimension.

Batch size determines how much of the training set the learning algorithm will go through, before updating its internal model parameters. The number of epochs determines how many times the learning algorithm will go through the whole dataset. If you have a batch size of 128 and the size of the dataset is 1024, the dataset can be equally divided into 8 batches. Then during a single epoch, it will train on these 8 batches one after another. If multiple epochs are chosen, it will then go to the next epoch with new batches [16].

To understand the learning rate, we will explain it in the context of gradient descent.

Gradient descent is used when you want to optimize the loss function. The basic principle behind gradient descent is that you want to find a local or global minimum. In relation to machine learning, it is often not possible to find the global minimum, since this requires that the loss function is convex, which is often not the case. Therefore you aim to find a local minimum which is "good enough". To do this you have to find the right learning rate. A too large learning rate could skip over the local minimum and will cause it to converge around the local minimum, but never reach it. A too small learning rate will take a lot of computations, causing the training to be slow [17].

4.2 Embeddings

Embeddings are a way to represent higher-dimensional vectors in a lower-dimensional form. It is used to represent things such as words, documents or other types of data as numerical vectors. These embeddings are learned from the data using a neural network. They capture the semantic or contextual relationships between words or items in the data [18].

One of the best-known models for making word embeddings is the Word2Vec model. This model aims to capture the meaning of words based on their co-occurrence patterns in the data. To better understand the concept of embeddings, we will now outline how Word2Vec specifically works [19].

The Word2Vec model

To use the Word2Vec model, a text corpus is needed. This is a collection of text data, which the model is to be trained on. This text corpus is then preprocessed, by splitting the text up into words, removing stop words such as "the", "is" and "and", and using other text normalization techniques such as stemming and lemmatization, which consists of removing prefixes and suffixes or reducing words to their root.

Next, a vocabulary is built from the preprocessed text corpus, where the words are mapped to unique integer ID's. Then input-output pairs are created, where the input is the word itself and the output is a neighbouring word within a fixed window size. This window size determines how close the words need to appear together.

Now the model is ready to be trained. The input layer consists of the one-hot encoding of the input word. A one-hot encoding is used for categorical variables, such as words. You could assign each word an integer value, but this suggests that there is an ordinal relationship between the words, when this is not the case. For instance, you could assign values to words such that "red" is 1, "blue" is 2 and "green" is 3. This would create the before mentioned problem. Instead, you use one-hot encoding, where you need 1 binary variable for each category. In the case of the colours, the binary variable would be equal to 1 for the specified colour and 0 for all the others. The encoding would look like this: $\begin{array}{ccc} \mathit{red} & \mathit{blue} & \mathit{green} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}$. This input layer then goes through the hidden layer, where all the different weights are learned and stored, and finally, it goes to the output layer, which typically uses a softmax activation function, which outputs probabilities of a word being related to another word.

When the model is trained, the weights of the input

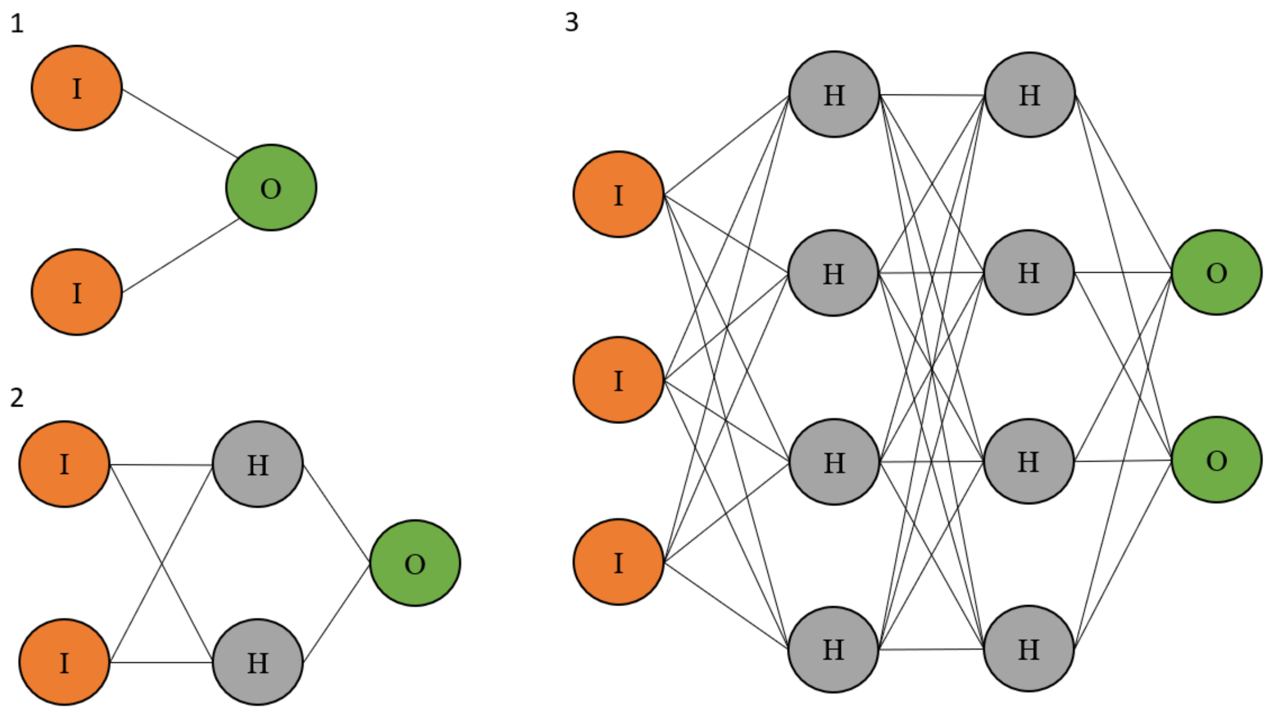

Figure 1: Three figures showing different types of neural networks. 1 = Perceptron, 2 = Feed Forward and 3 = Deep Feed-forward

layer are updated to learn the word embeddings. The embedding for each word is the set of weights that connects the input layer to the hidden layer. When the model is trained, the embeddings should be able to accurately predict the context of a word based on its neighbouring words [20][19].

Another thing, which can be the goal of these embeddings, is the classic example of these four vectors and their relationship, which should hold true: $king - man + woman = queen$ [21].

Now that this model has been briefly described, and there is some basic understanding of embeddings, we can move on to graph embeddings.

Embedding of graphs

When embedding graphs, there exist different kinds depending on your goal with the embeddings. Vertex embeddings describe the connectivity of nodes, path embeddings are traversals across the graph, and graph embeddings encode an entire graph into a single vector.

As mentioned in Section 4.1, there are many different hyper-parameters, one of them being the embedding dimensions. Embedding dimensions describes how many dimensions the final embedding vectors exist in. This value can be anything, but the common embedding model Word2Vec which is described in section 4.2, usually uses 100, 300 or even more. When choosing the size of the embedding dimension, things such as the specific task, the complexity of the data and the available computational resources should be taken into account [22].

In these next sections, we will go over some different graph embedding models, that we plan to use in this project.

The TransE model

The general idea behind the TransE model is to embed the entities and relations such that if a triple, (h, r, t) = (head, relation, tail) exists, then the vector created by the embedding for the tail entity should be close to the head entity vector plus the relation vector. Since only a single low-dimensional vector is created for each entity and each relation, it is a matter of minimizing the distance between these vectors mentioned above. To enhance the model's ability to distinguish between true triples and false triples, and to be able to predict new triples more accurately, a new form of triples called corrupt triples is made. These triples are created from the true triples, by replacing either the head entity or the tail entity with a random entity from the graph. This is what is meant when we talk about corrupting triples. The goal of these corrupted triples is to maximize the distance between the head entity vector plus the relation vector and the tail entity vector [23].

The ComplEx model

The ComplEx model is quite similar to other vector-based embedding models. The main difference is that the ComplEx model uses vectors with complex numbers, instead of vectors with real numbers. This allows the embedding to capture rich interactions between entities and relationships. The reason for this is that when you work with real vectors, usually the standard dot product is used, which is symmetric, meaning that if we have the real vectors A and B then $A \cdot B = B \cdot A$. When calculating the dot product for complex vectors, the Hermitian dot product is used instead. This calculation is not symmetric, because it involves the complex conjugate, which is only used for the second vector. The complex conjugate of a complex number is the imaginary part of the complex number negated. So if we got a complex number $a + b \cdot i$ then its complex conjugate would be $a - b \cdot i$. This means if we got complex vectors C and D then $H(C, D) \neq H(D, C)$ where H is the function for the Hermitian dot product. This asymmetry allows different scores for facts about asymmetric relations, which are relations where the order matters. Even though the Hermitian dot product is asymmetric it still retains the efficiency benefits of the standard dot product, which is linear in both space and time complexity [24].

The ConvKB model

The ConvKB model makes use of a convolutional neural network (CNN), which is commonly used for image recognition tasks, but in this case, it is adapted to work with graph data. The goal of the ConvKB model is to make link predictions. It takes the triples of the knowledge graph as input and converts these into a numerical representation. It then moves a filter across the triples, which captures local patterns by looking at neighbouring triples. The CNN then applies a mathematical operation to these triples to extract features or useful representations. Then it refines the representation further, by using a fully connected layer, which takes the learned features and maps them to a lower dimensional space. Lastly in the output layer, it predicts the missing relationships [25].

4.3 Dataset partitioning

When doing machine learning it is important to split the data into multiple parts. These parts consist of a train, test and validation set. The key purpose of the test set is to have some data which the model has never seen or interacted with to measure how well the model generalizes to new data. The validation set's purpose is to have a set where different settings for the hyperparameters can be tested and find the optimal hyperparameters [26].

Regularization techniques are a form of hyperparameters, where the goal of it is to avoid overfitting. Two common techniques for this are called Lasso and Ridge regularization [27].

The validation part is data that the model sees when tuning the hyperparameters but does not explicitly learn from. Meaning that the model evaluates the performance using different hyperparameters on the validation set, and from this, the best out of the tested hyperparameters can be chosen.

This works by testing different hyperparameters, such as the ones mentioned in Section 4.1. There exist different approaches to this tuning, one of these being a grid search. In an exhaustive grid search, the hyperparameters are split up into different discrete values, as seen in figure 2, where every combination is tested. If you have more hyperparameters than 2, you can imagine more dimensions. This can be very time-consuming as the model needs to be trained for every test of a new combination of hyperparameters. After each training the model is then evaluated on the validation set, to see the effect of the hyperparameters. To make it less time-consuming, a random search can be conducted, where instead a random subset of the hyperparameters in the grid search is tested, this can be seen in figure 2. The larger this subset, the better hyperparameters it is likely to find, but it will also take longer. The smaller the subset, the faster it is, but it might not find the best hyperparameters. In the end, you hopefully end up with a reasonably good model, depending on how many combinations of hyperparameters you tested [28].

Graph specific considerations

When splitting the data into three parts for graphs, there are some extra considerations to take into account. The first of these is deciding whether it is wanted to do inductive or transductive link prediction. In inductive link prediction for graphs, there are entities and/or relations in the test and validation set, which are not in the testing set. This means that the model also has to predict these new entities and relations and their connections, even though it has never seen them before. In transductive link prediction, all of the entities and the relation types that are in the testing set and the validation set are also in the training set. This means that the model does not have to predict these new entities or relations, but only the missing links between them [29].

Another important thing to take into consideration is if the graph contains multiple edges between the same node. For instance, if we have the edges Copenhagen IsTheCapitalOf Denmark and Denmark hasCapital Copenhagen, then these edges should be placed in the same set. The reason for this is, that it is not a very interesting fact to learn that Copenhagen is the capital of Denmark if we already know that Denmark's capital is Copenhagen. Therefore these should not be placed in different sets, since we do not want to evaluate the model's performance based on its ability to predict these simple relationships [30].

The case where the same edge combination is in two different sets is called data leakage, this can lead to over-optimistic performance estimates. Thereby the model looks better than it actually is. The aftereffect will often be an unpleasant surprise when the model does not perform well on new data without data leakage [31] [32].

5 Data analysis and preparation

In Section 5.1 we will first describe each dataset, respectively the BioNev data, CTD_DDA and NDFRT_DDA and the PharmKG8k dataset. The descrip-

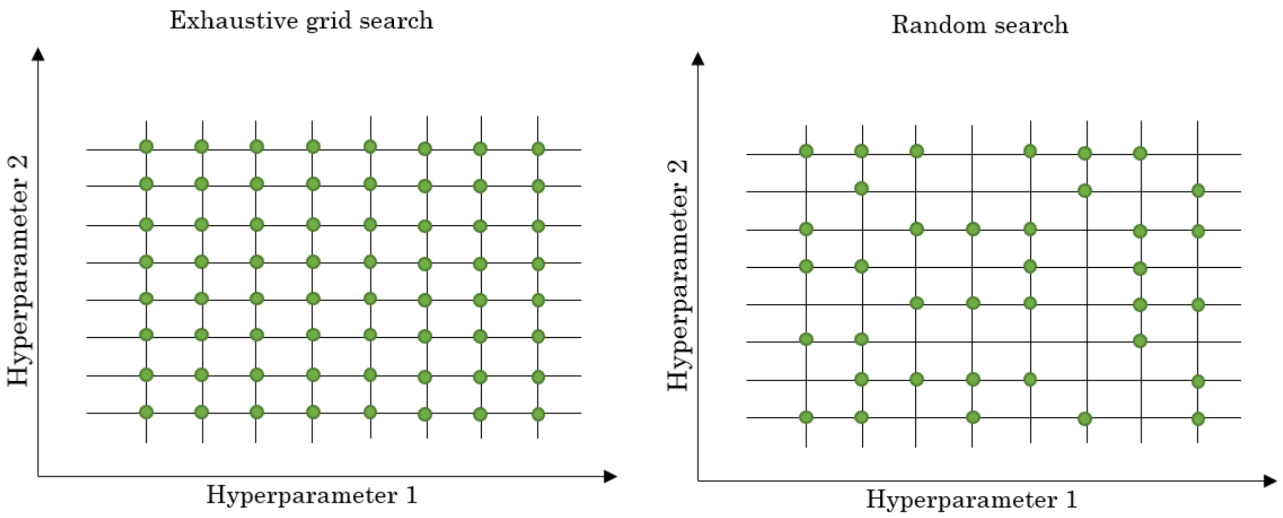

Figure 2: Two figures, the first one showing the chosen hyperparameters for an exhaustive grid search, the second one showing the chosen hyperparameters for a random search. Each line represents a discrete value for a hyperparameter. Each dot represents a combination of two hyperparameters' discrete values.

tion will include different statistics such as the number of entities and edges in each dataset. Lastly, will we describe the UMLS dataset, which is used to align the dataset, with a common ID. Afterwards, in Section 5.2, we will describe how we have used the UMLS dataset, to align the three datasets. In Section 5.3 the three datasets will be compared. The comparison will include an overview of matching entities and edges and a semantic types comparison, made by using an API search in the UMLS database. Lastly, in Section 5.4 an improvement of NDFRT_DDA will be made based on the findings in Section 5.3.

5.1 Preliminary data analysis

First, we will describe the datasets dealing with drug-disease associations, these are the CTD_DDA and NDFRT_DDA datasets, which were taken from the GitHub page that belongs to the paper "Graph embedding on biomedical networks: methods, applications and evaluations" [7], this will be done in Section 5.1

Then in Section 5.1 the dataset called PharmKG8k, which has been found from the Pykeen package in Python [6], will be described. The section will also include an explanation of the issues related to the PharmKG8k split into training, validation and testing sets.

The description of CTD_DDA, NDFRT_DDA and PharmKG8k, will include an overview of the number of entities, relations, and triples in each dataset and include different statistics related to each dataset.

Then in Section 5.1 the UMLS data will be described. That dataset is used to map the entities in PharmKG8k, to a common ID, called the *UMLS_CUI*, that CTD_DDA and NDFRT_DDA already have.

BioNev data

The data has been collected from the GitHub of an earlier research paper *Graph embedding on biomedical networks: methods, applications, and evaluations* [7].

The data on the Github consists of 7 files `CTD_DDA`, `Clin_Term_COOC`, `DrugBank_DDI`, `Mashup_PPI`, `NDFRT_DDA`, `STRING_PPI` and `node2vec_PPI`, and some subfiles.

For the purpose of the project, the focal point will be on the datasets CTD_DDA and NDFRT_DDA.

For keeping the names of the files short two different abbreviations are used. CTD is an abbreviation for *Comparative Toxicogenomics Database* [33], NDFRT is an abbreviation for *National Drug File Reference Terminology* [34], these datasets data are collected from the same place, from which their abbreviations are from. DDA is an abbreviation for *drug-disease association* [7]. These abbreviations can also be seen in Table 1. The abbreviations will be used throughout the report.

CTD_DDA and NDFRT_DDA each have 2 subfiles and each dataset is under the category of link prediction.

In CTD_DDA and NDFRT_DDA there are two subfiles consisting of an edgelist containing a list of pairwise indexes for entities, which signifies an edge between them. Next, there is a node_list, which contains the different entities with their different IDs, respectively index, CTD_ID and UMLS_CUI. Lastly, the file contains a type value that indicates whether the node is a chemical or a disease. There are 12,765 entities and 92,813 edges in CTD_DDA, in NDFRT_DDA there are 13,545 node and 56,515 edges-

Through an API search based on the UMLS_CUI IDs in the UMLS database we have found out that,

Table 1: Abbreviations in datasets and through the report

Abbreviations	Meaning
CTD	Comparative Toxicogenomics Database
NDFRT	National Drug File Reference Terminology
DDA	drug-disease association

CTD_DDA entities have 132 different semantic types combinations, its most common semantic type combination is; *"Organic Chemical", and "Pharmacologic Substance"*, which make up 33.57% of the datasets entities. 32 of the semantic combinations are only present once. NDFRT_DDA has 90 different semantic type combinations, its most common semantic type combination is; *"Clinical Drug"*, which makes up 64.10% of the datasets entities. 25 of the semantic combinations are only present once. An overview of the different semantic types can be seen in Table 4. A comparison of the semantic types will be elaborated in Section 5.3.

In Table 2 an overview of the number of edges, entities, and relations in each dataset, and the density can be seen. Density describes how many edges there are, compared to how many there could be if every node were connected. Furthermore, the connected component, the minima degree, maxima degree, median degree and average degree can be seen, as well as the number of entities with 1 degree.

PharmKG8k data

The PharmKG8k knowledge graph consists of 7,247 entities, 28 relations and 485,787 triples. The relations have 4 super categories, which consist of interactions, disease-gene, disease-chemical and chemical-gene. The more specific meanings of each relation can be found on page 6 in the PharmKG paper [12]. The knowledge graph is one of Pykeens. Pykeen [6] is a package for Python which already has some different knowledge graph datasets and also has built-in methods for machine learning with various different models. The original knowledge graph was made in the context of the paper *PharmKG: a dedicated knowledge graph benchmark for biomedical data mining* [12], in their paper, they state that they have 7,603 entities, 500,958 edges and 29 relations. On the Pykeen GitHub they state, that they have 7,247 entities, 485,787 edges and 28 relations. Nevertheless, what we have found is that the PharmKG8k dataset contains 7,262 entities, 485,787 edges and 28 relations.

We made an issue on Pykeen's GitHub, and they could not give an answer to why the inconsistencies exist. Thereby, we can only guess what has caused the inconsistencies, and therefore we will just state that the duplicates will not have an unintended impact. We will, among other things, clean PharmKG8k in Section 5.2, so that we also have the data without the duplicates. Through the issue, we were sent another GitHub which contains the data Pykeen uses [35].

The PharmKG8k is constructed based on six different public databases OMIM, DrugBank, PharmGKB, Therapeutic Target Database (TTD), SIDER and HumanNet [12].

The PharmKG8k that is collected from the GitHub biomed-AI/PharmKG only have the data where it is split up into train, test and valid like pykeen. These sets both have external and internal duplicates, which means that there are duplicates internally in the sets and between all of the sets. For instance, for the train set, there are 14,020 duplicates and 2,913 duplicates between the train and test set. This is especially a problem when trying to make link predictions on the data, since the same edges in, for example, train and test, will lead to misleadingly good test results, so the testing scores will not be representative of how the model will perform on new data. This problem will be dealt with when all the data is collected from CTD_DDA, NDFRT_DDA and PharmKG8k, here we want to make sure that there are no duplicates, either external or internal between train, test and validate.

An overview of the number of entities and edges as well as other Statistics related to PharmKG8k can be seen en 2

UMLS data

The UMLS data is from *National Library of Medicine* the specific data used is `MRCONSO.RRF`. `MRCONSO.RRF` was published on November 7, 2022. The dataset consists of multiple ID's including the ID *UMLS_CUI*, which is also in the `CDT_DDA` and `NDFRT_DDA` knowledge graphs. The dataset includes the names of medicines and diseases in different languages and synonyms within each language. To get access to the data you have to make an application, after that the data can be found on the *National Library of Medicine* website [36].

5.2 Data alignment

The goal of preparing the data is to enable a merge of the datasets `CTD_DDA` and `NDFRT_DDA` and `PharmKG8k`. The preparation includes making a common ID, to ensure we can merge the datasets correctly.

In the BioNev datasets, there is an ID called UMLS_CUI, which is from the Unified Medical Language System [36]. There is no such ID in the PharmKG8k dataset, only an attribute called label, where the name of the disease, medicine, gene, etc. is contained. So to be able to connect these two datasets,

Table 2: Information about statistics for each dataset.

Datasets	CTD_DDA	NDFRT_DDA	PharmKG8k
Number of entities	12,765	13,545	7,166
Number of edges	92,813	56,515	414,018
Density	0.06%	0.03%	0.8%
Weakly connected components	20	85	1
Min degree	1	1	1
Max degree	1,217	845	4,627
Median degree	3	3	64.0
Average degree	14.54	8.34	115.55
entities with 1 degree	4,036	3,397	55

it was necessary to find the UMLS_CUI matching the labels in the PharmKG8k dataset.

It was possible to find a dataset with the names of the different medical terms and the accompanying UMLS_CUI on the UMLS website [36], the UMLS_CUI dataset is described in Section 5.1.

The UMLS dataset has multiple spellings and synonyms of each medical term in different languages. So first, all other entities than the ones in English were removed.

Next, there were multiple unusable attributes, for instance, various different kinds of ID's. We removed these ID's, so the only two left were UMLS_CUI and *name*. The name attribute sometimes had capitalized letters and multiple different special characters. The PharmKG8k dataset had neither of these. So we removed the special characters and decapitalized the letters.

After this was done we made an intersection between the UMLS dataset and the entities in the PharmKG8k dataset.

We also found out by doing these matches, that some of the names in the PharmKG8k actually are synonyms. There were 137 of these which were synonyms of something else, out of these there were 67 pairs of names which got matched to the same UMLS_CUI, and then the last 3 were matched to the same UMLS_CUI. Unfortunately, there were 27 names which we did not match to a UMLS_CUI. So out of the 7,262 entities in PharmKG8k, we ended up matching 7,235 of the entities to 7,166 different UMLS_CUI's. Because of these 27 removed entities, we also lose 864 edges from the original training set, and 119 edges from both the validation and test set.

There is just one last problem and that is that the NDFRT_DDA and CTD_DDA datasets do not contain any relation types. There are two obvious solutions to this. One is to add a relation type to these, and the other is to remove them from PharmKG8k. Since we do not want to lose data, we have decided to add the relation types NDFRT and CTD to them respectively. These would then represent that these edges come from these datasets. Unfortunately, we could not find any already existing relation types in PharmKG8k which would adequately explain these edges, so this was in our opinion the best alternative.

Now the PharmKG8k dataset also contained UMLS_CUI, and the CTD_DDA and NDFRT_DDA datasets contained a relation type, it was now possible to join them.

5.3 Comparative graph analysis

In this section, the datasets PharmKG8k, CTD_DDA, and NDFRT_DDA will be compared. This is done to get an overview of the different datasets and will later be useful for dividing the datasets into train, test, and validation sets.

In table 3 matches between the different datasets can be seen. Both matches of entities and edges can be seen. The proportion of matches is calculated both for the Jaccard similarity and the overlap coefficient. The Jaccard similarity is calculated by dividing the inner join by the number of unique entities/edges. For instance, for the matches of entities between CTD_DDA and NDFRT_DDA, CTD_DDA has 12765 entities, NDFRT_DDA has 13454 and there are 2466 matches between them. This means that there are $12765 + 13454 - 2466 = 23844$ unique entities, so this is the number that is divided by. The overlap coefficient is calculated by dividing the inner join by the size of the smallest set. If we use the same example as before the calculation would be $2466/12765 = 19.32\%$. Since the three datasets have different relation types, we do not consider this, when comparing the matches of edges between the datasets. In addition, these comparisons are only head-tail to head-tail.

By table 3, it is clear that all the datasets are quite different. Even though `PharmKG8k` has a lot of edges compared to the number of entities, there are not

Table 3: Proportion of matching entities and edges, between the different datasets.

Datasets	CTD / NDFRT	CTD / PharmKG8k	NDFRT / PharmKG8k
Jaccard similarity:			
Matching entities	2,466/23,844 = 10.34%	1,950/18,062 = 10.80%	1,543/19,249 = 8.02%
Matching edges	1,219/148,109 = 0.82%	4,902/501,929 = 0.98%	211/470,322 = 0.04%
Overlap coefficient:			
Matching entities	2,466/12,765 = 19.32%	1,950/7,166 = 27.21%	1,543/7,166=21.53%
Matching edges	1,219/565,15 = 2.16%	4,902/928,13 = 5.28%	211/565,15 = 0.37%

many matching edges between `PharmKG8k` and `CTD_DDA` and `NDFRT_DDA`. `CTD_DDA` and `PharmKG8k` are those that match best, afterwards, `NDFRT_DDA` and `CTD_DDA`, match almost as well as `CTD_DDA` and `PharmKG8k`. `NDFRT_DDA` and `PharmKG8k` match's is the worst.

Since each knowledge graph has something in common, but not everything in common, it may make sense to combine the knowledge graphs into one and then check whether this combined knowledge graph can make better predictions than the knowledge graphs individually.

Semantic comparison

To get a further understanding of the dataset's differences and similarities, we will use UMLS_CUI to check the semantics of each entity on the *Natural Library of Medicine* [36].

An entity can have more than one semantic type, so to fully check the similarities, we have chosen to have each category as a combination of the semantic types. In the appendix, section 11.1 5 tables can be seen to get an overview of the semantics in each dataset, and which semantics the datasets have in common.

In Table 4 we have selected some interesting observations from the tables in Section 11.1. It turns out that the most common semantic in NDFRT_DDA is "Clinical Drug", 64.10% of NDFRT_DDA data classifies as "Clinical Drug". The semantic "Clinical Drug" is in neither CTD_DDA nor PharmKG8k. However, the most common semantic type for CTD_DDA is the combination of "Organic Chemical", "Pharmacologic Substance", which constitutes 33.57% of CTD_DDA data, while the combination of "Organic Chemical", "Pharmacologic Substance" is around 14% for both NDFRT_DDA and PharmKG8k. The dataset PharmKG8k's most common semantic type is "Gene or genome", which constitutes 52.94% of PharmKG8k's data. The semantic "Gene or genome" is in neither CTD_DDA nor NDFRT_DDA.

In a certain way, CTD_DDA differs from NDFRT_DAA and PharmKG8k by not having a semantic type, which is a large part of the data, where this semantic type is neither in NDFRT_DAA nor PharmKG8k. The semantic type that CTD_DDA have the most, where NDFRT_DDA and PharmKG8k have none is the combination of "Hazardous or Poisonous Substance", "Inorganic Chemical". However, this combination only contributes to 0.36% of CTD_DDA's data.

It should be noted that "Pharmacologic Substance" is closely related to "Clinical drug". While a pharmacologic substance denotes the active ingredient in a drug, a clinical drug is what the specific drug is, sometimes even being as specific as also including the dosage and route of administration. One example of this in the UMLS database would be with the pharmacologic substance troglitazone [37]. The semantic type of this is both "Organic chemical" and "Pharmacologic Substance". You can then look at the entry of "troglitazone 200 MG Oral Tablet" [38], instead which is more specific containing both the dosage and the fact that it is an oral tablet. The semantic type of this is "Clinical drug". So two very closely related things can be placed in these different semantic type categories.

5.4 Data augmentation

Since we can see that NDFRT_DDA is the only dataset containing entities which only have the semantic type clinical drug and that it is possible to match at least some of these to a broader concept, as in the example mentioned in Section 5.3, we want to make new entities and edges in NDFRT_DDA, to improve the connectivity between NDFRT_DDA and the other datasets. The way we want to do this is by using the API for the UMLS database [39]. After investigating the names in the UMLS database for the drugs with the semantic type clinical drug, we found that many of these contain the name of the active ingredient followed by the dosage in MG. So we decided to look for the string "MG" in all of the names of the clinical drugs in NDFRT_DDA, and then extract the words which come before "MG" and then do an API search for this, to find the UMLS_CUI for this active ingredient.

In the entities of NDFRT_DDA, 12,377 is of the type "drug", and of these 8,724 is of the type "clinical drug". Using this method we were able to find new UMLS_CUI's for 7,710 out of the 8,724 "clinical drugs". We checked these 7,710 and found out that 163 of these, contained more than one active ingredient, so naturally, we couldn't match these to a single

Table 4: Interesting differences in the semantic types, of the three different datasets entities

Semantic types	NDFRT_DDA	CTD_DDA	PharmKG8k
"Clinical Drug"	64.10%	0.00%	0.00%
"Gene or genome"	0.00%	0.00%	52.94%
"Hazardous or Poisonous Substance", "Inorganic Chemical"	0.00%	0.36%	0.00%
"Organic Chemical", "Pharmacologic Substance"	14.17%	33.57%	14.97%
"Disease or Syndrome"	5.90%	15.11%	9.64%
"Organic Chemical"	0.18%	14.63%	0.21%
"Organic Chemical", "Hazardous or Poisonous Substance"	0.04%	3.48%	0.08%
Others	15.61%	32.85%	22.16%

UMLS_CUI so we decided not to match these. There also were some other errors in the 7,710 clinical drugs, such as the search term we were using had the prefix "ML" or "HR" at the beginning of the string. This error was in 667 of them. This caused it to match to a more specific drug than wanted. So these prefixes were removed and a new search was made. In 17 of the names of the clinical drugs, "MG" was in the string, but none of these returned a result. We were able to manually go through them and find matches for 5 of these. So in the end we ended up matching 7,552 of the 8,724 (86.57%) clinical drugs to a more general drug.

Since we can't know with 100% certainty that the correct ones have been found, we have decided to manually check a sample of these for correctness. We have chosen a sample size of 366, which is the sample size needed for a population size of 7,552 for a 95% confidence level and a 5% margin of error.

Out of the sample, we found 7 errors. The error was due to abbreviations for chemicals such as NA, SO4, and HCL instead of sodium, sulfate and hydrochloride. There were also problems with the word ACTUAT appearing, which is not a word we want to search for because it is used in the context of a device which has multiple dosages to deliver, such as an inhaler. ACTUAT signifies how many dosages a device has, or the dosage per ACTUAT. Some of the problems were also due to the number of words before the string "MG" since some names had more than 3 words before "MG". We choose to use 3 words before "MG" since this looked like the most common solution, using more than 3 words, would have more errors because words like "HR", "ACTUAT" and "ML" often would be the fourth word.

These different problems were fixed, and we took a new sample of the same size. Unfortunately, there were 8 errors found. Though it should be mentioned that these errors were not drastic, as they did not link to the completely wrong thing. Sometimes it just linked to another just as specific thing as before and sometimes a too broad concept. An example of a specific thing being connected to another but slightly different thing is, *VANCOMYCIN HCL 7MG/ML/NACL 0.9% INJ,BAG,250ML* and the new thing *vancomycin 125 MG Oral Capsule*. These are still closely related but are not exactly what we want. It was wanted that it should be connected to either *vancomycin* or *vancomycin hydrochloride*. An example of a thing being connected to a too broad concept would be *house dust allergenic extract 400 MG/ML Injectable Solution* and *Allergenic extract*. In this case, the wanted connection would have been *house dust allergenic extract*.

Due to time constraints, we could not fix these different errors. Instead what we decided to do was remove any edges of the newly added, where the node of the new connection did not exist in either of the three datasets. This made sure, that the density of the merged final graph would be higher, than without these new edges. It is also likely that at least some of these removed edges were edges which made a less than ideal connection.

There were 82 entities which did not exist in either of the three datasets. These 82 entities were present in 172 out of the 7552 edges, so in the end, 7380 edges were added to the NDFRT_DDA dataset, because none of these edges was already in the dataset. Since 26 of the entities did not exist in NDFRT_DDA but existed in one of the other datasets, 26 entities were also added.

To keep the original meaning of the relation types, as mentioned in Section 5.2 we decided to add a new relation type to these edges, named subgroupOf.

5.5 Analysis of the original splits

To get insights on how the PharmKG8k, CTD_DDA and NDFRT_DDA datasets previously have been split up into a train, test and validation set, and to further investigate the duplicates in PharmKG8k discovered in Section 5, we will in this section give an overview of different statistics describing the splits. But first, we will describe how each dataset was originally split up.

The datasets CTD_DDA and NDFRT_DDA have, as mentioned in Section 5.1, been used in the making of the same paper, and therefore the same split technique has been used on both of them.

The split technique is straightforward since they randomly split the data into only 2 groups; train and test. Where the proportion of the train set is 80% and the proportion of the test set is 20%. Since each dataset has no entity-to-entity duplicates, meaning that if we have an edge (e_1 *to* e_2), there exist no edge (e_2 *to* e_1) or no other (e_1 *to* e_2).

The split technique used on the CTD_DDA and NDFRT_DDA datasets does not have to take data leakage into account when making the split. This is because it is impossible to have any data leakage since that would be due to entity-to-entity duplicates, which are as mentioned not present in the datasets. However, this split will not satisfy the condition for a transductive split, since it can't be insured that all entities in the test set also are in the train set.

The original PharmKG8k split was split into 80% training and 10% for both validation and testing. We do not know how the PharmKG8k have been split, but it does not look like data leakage and unseen entities in the test and validation set have been a priority. The split had entity-to-entity duplicates between all sets, and all entities in the validation and test sets were not present in the train set, thereby an inductive split with data leakage. The mentioned entity-to-entity duplicates are both of the cases where we have (e_1 *to* e_2) and then another edge also (e_1 *to* e_2) but with a different relation type. There also exist duplicates where (e_1 *to* e_2) and (e_2 *to* e_1), have the same relation types, so by inverting the edge, we get the exact same edge again.

In the following, we will mention what we call head relation tail, where head and tail are two entities.

The head corresponds to the source entities and the tail corresponds to the target entity. When we write regular duplicates it means an edge duplicate where $head_1 = head_2 \wedge tail_1 = tail_2$, and when we write inverse duplicates it means an edge duplicate where $head_1 = tail_2 \wedge tail_1 = head_2$.

PharmKG8k split

In PharmKG8k's train set, there are 7,247 different entities, in the test set there are 6,724 and in the validation set, there are 6,689. It has 500,360 edges in total, where 400,788 are in train corresponding to $\approx$ 80.10%. There are 50,036 edges in the test set corresponding to 10% and there are 49,536 edges in validation corresponding to $\approx$ 9.9%. An overview of the number of entities and edges in each set can be seen in Table 5.

PharmKG8k is the only dataset with data leakage in its split sets. Where data leakage means that the combination of two entities is present in more than one set. An example could be having two edges where $head_1 = head_2$ and $tail_1 = tail_2$ or $head_1 = tail_2$ and $tail_2 = head_1$. The PharmKG8k split has 60,490 edges causing data leakage in total, 22,999 of these are regular duplicates and 38,191 are inverse duplicates. What specific sets the overlapping edges are between, can be seen in Table 6, together with the total number of regular duplicates and inverse duplicates. In addition in each set, there are internal duplicates, in the train set, there are 14,020 duplicates. In the test set, there are 265 duplicates and in the validation set, there are 269 duplicates.

Because of the severe data leakage and the relatively high density in the graph, the split only has 6 unseen entities in the test set and 11 in the validation set. Where an unseen entity signifies an entity that is present in the test or validation set but not in the train set. The unseen entities in the test set are present in 7 edges in the test set and the unseen entities in the validation set are present in 12 of the edges.

In addition to the many regular and inverse duplicates, there are also 100% where it is not just the head and tail that match, but the entire edge, these are equivalent to the internal duplicates but are between different sets. It should be noted that these 100% duplicates are not a subset of the regular duplicates, since these get removed before checking the regular duplicates. Between the train set and the test set, there are 2,913 100% duplicates. Between the train and validation set, there are 2,771 100% duplicates, and between the test and validation set, there are 501 100% duplicates, an overview of these 100% duplicates can be seen in Table 6.

CTD_DDA and NDFRT_DDA splits

In the CTD_DDA train set, there are 11,857 different entities and in the test set, there are 6,924. It has 92.813 edges in total, where 74,251 are in the train set corresponding to $\approx$ 80%, and there are 18,562 edges in the test set corresponding to $\approx$ 20%. An overview of the number of entities and edges in each set can be seen in Table 5.

NDFRT_DDA has 12,786 different entities in the train set and in the test set, there are 7,322. It has 56,515 edges in total, where 45,212 are in the train set corresponding to 80%, and there are 11,303 edges in the test set corresponding to 20%.

Both the CTD_DDA and NDFRT_DDA datasets can not have any data leakage, since all the entity combinations there exist only exist once in the whole graph. However, each dataset has a lot of unseen entities in their respective test sets. CTD_DDA has 908 unseen entities in the test set, and 992 edges in the test set contain these unseen entities. NDFRT_DDA has 759 unseen entities, and 868 edges in the test set contain these unseen entities.

Table 5: Number of entities, edges and internal duplicates - The internal duplicates are full head relation tail duplicates.

Dataset	Set	Number of entities	Number of edges
PharmKG8k	Train	7,247	400,788
	Test	6,724	50,036
	Validate	6,689	49,536
CTD_DDA	Train	11,857	74,251
	Test	6,924	18,562
NDFRT_DDA	Train	12,786	45,212
	Test	7,322	11,303

Table 6: Data leakage in PharmKG8k

Dataset	Set	Regular duplicates	Inverse duplicates	100% duplicates
PharmKG8k	Train/Test	10,336	17,750	2,913
	Train/Validation	10,160	17,555	2,771
	Test/Validation	1,803	2,886	501
	Total	22,299	38,191	6,185

6 Data cleaning and normalization

As we found out in Section 5.5 there are some significant problems with the PharmKG8k original split.

Because of the lack of domain knowledge, we did not anticipate, that those kinds of problems even could exist. Even if we would have had that knowledge, we would not have expected these problems in a standard published dataset, such as the PharmKG8k dataset which is used in the Pykeen data library for transductive link prediction.

We thought that it would be interesting to quantify the effect of issues in the splits on the estimate prediction quality. Therefore we have decided to make 3 different splits of the PharmKG8k data. These three are as follows, a transductive with data leakage, a transductive without data leakage and an inductive without data leakage. These three splits allow us to compare the effect of data leakage and the effect of an inductive split, compared with the properly made split, a transductive without data leakage. It might sound odd to make inductive splits, since these models that we are going to use only work with transductive splits, and are not able to predict unseen entities or relation types. But the Pykeen package is able to handle inductive splits of datasets, by removing those edges from the test and validation splits, thereby making them transductive but with less data and smaller test and validation sets. This means that when we mention inductive splits from now on, we mean that they start off as inductive, but Pykeen handles this and changes them to transductive by the before mentioned method and thus the test or validation sets are effectively downsampled.

When we say that a dataset is without data leakage it can be explained mathematically where we will use these denotations:

$$(h, r, t) = (head, relation, tail)$$

$$D_{tr} = \textit{the train set}$$

$$D_{ts} = \textit{the test set}$$

$$D_{va} = \textit{the validation set}$$

$$D_{\omega} = \textit{the full dataset}$$

And this is the relationship between these:

$$D_{tr} \cup D_{ts} \cup D_{va} = D_{\omega}$$

Then a graph dataset having no data leakage can be expressed as this holding true:

$$\begin{aligned} &\forall(h, r, t) \in D_{tr} : \neg(\exists(h', t') \in (D_{ts} \cup D_{va}) : \\ &\qquad ((h = h' \land t = t') \lor (h = t' \land t = h'))) \\ &\qquad\qquad and \\ &\forall(h, r, t) \in D_{ts} : \neg(\exists(h', t') \in (D_{tr} \cup D_{va}) : \\ &\qquad ((h = h' \land t = t') \lor (h = t' \land t = h'))) \\ &\qquad\qquad and \\ &\forall(h, r, t) \in D_{va} : \neg(\exists(h', t') \in (D_{tr} \cup D_{ts}) : \\ &\qquad ((h = h' \land t = t') \lor (h = t' \land t = h'))) \end{aligned}$$

This means that no pair of nodes that exists in the train set, should exist in either the test or validation set. It should neither appear in the regular order nor inverted. Vice versa for the test set, and the validation set. If the dataset then instead has data leakage, it would mean that this mathematical expression does not hold true, since there then would exist a pair of nodes, for instance in the train set, which also exists in for instance the test set in the regular order or inverted.

6.1 Transductive split with data leakage

In the original PharmKG8k, and as mentioned in Section 5.5 there were only 17 entities and 19 edges in the test and validation set, which caused the split to be inductive.

In Section 5.2, we added the UMLS_CUI to the PharmKG8k original split, and here we reduced the number of entities, both by matching some of them to the same UMLS_CUI and because there were 27 which we could not find. All in all the number of entities gets reduced by 96. So both because of our matching of the synonyms and the removal of the 27 entities, we by chance end up changing or removing the entities which caused the split to be inductive before. Therefore by just adding the UMLS_CUI to the possible entities we got a transductive split with data leakage.

Compared to the original PharmKG8k split, this split has more data leakage. This is the case since we have merged some of the entities in PharmKG8k into a single entity, which apparently has a bigger effect on the data leakage than removing some of the entities, thereby removing some edges. For instance, the total amount of data leakage between the train and test set now amounts to 28355 intersects, compared to the previous 28086 intersects found in Section 5.5.

6.2 Inductive split without data leakage

This split is made to access how the loss of edges, due to downsampling in Pykeen when an inductive split is used in their transductive library, can affect the results of the model.

To make this split, we start off with the split made in Section 6.1, we then input these train, test and validation sets and append them together. This dataframe gets looped through and we move each row into the new training set. For each row, we move we then check if another row with the same "head, tail" or "tail, head" combination exists and also move these rows into the new training dataframe. This is done to ensure that we do not have any data leakage in the final split. After this is done for a row, we add the "head, tail" and "tail, head" to a moved_pairs set. Then for each new row, we check if they are in the moved_pairs set, and skip over them if they are. Then when the size of the new training dataframe is at least 80% of the original PharmKG8k dataset, we break out of this loop and the new train set dataframe is made.

To then make the test set, this process is repeated, but instead of looping through the dataframe with all data, the new train set is removed and we loop through the PharmKG8k_no_train set instead.

To finally make the validation set, the test set is removed from the PharmKG8k_no_train set, and this is then the validation set.

It should be noted that this does not check if the split is transductive, so as expected we end up with an inductive split.

6.3 Transductive split without data leakage

This split is the only split which has been made for all of the datasets, NDFRT_DDA, CTD_DDA, PharmKG8k and the full dataset, which includes all of these.

The split for PharmKG8k is based upon the inductive split without data leakage in Section 6.2. To make this split transductive, we found all of the entities and relation types which only existed in the test or valid set and not in the train set. There were no relation types, that were in the test and validation set that were not in the train set. The edges with the entities that were not in the train set but were in the validation or test set, were moved from these sets into the train set.

To keep the split sizes intact, some edges from the train set, have to be moved back into the validation and test set respectively. To make sure that none of these edges creates data leakage, a dataframe is made that only contains the edges from the train set where each entity-to-entity combination only exists once.

Then we use our self-made function make_transductive() which starts off with counting the number of entities and saves them in a dataframe. Then the entities which appeared fewer times than 2, were removed from this dataframe. Whenever an edge then was wanted to be moved, it was checked if both head and tail were in the dataframe, and if it was, the edge was moved and the count of the entities in the dataframe was decreased by 1, and if the edge moved caused an entity count less than 2, that entity would be removed from the dataframe. By doing this it is made sure that whenever an edge is moved from the train set, there is still at least 1 other edge containing these entities. After moving edges like this from test to train, it then moves edges from validation to train.

When making the splits for NDFRT_DDA, we wanted to keep in mind that these splits would eventually be joined into the full dataset with CTD_DDA and PharmKG8k. Since we would like to be able to compare the NDFRT_DDA model results with the full data set, it was important that the NDFRT_DDA split made was transductive alone, but also could be joined with the train, validation and test set for PharmKG8k and CTD_DDA without causing data leakage. Therefore to avoid introducing data leakage into the full dataset split, we had to base the train, test and validation split for NDFRT_DDA on the PharmKG8k split. This means that we checked every head, tail and tail, head combination in the NDFRT_DDA dataset, with all of the head, tail combinations in the PharmKG8k training set. These would be in the NDFRT_DDA training set. The same was done for the NDFRT_DDA test set and validation set.

Then the rest of the edges were sampled into the train, test, and validation sets respectively, to get the correct split sizes, the sample took into account the edges that were already in the train test and validation sets. After that we had an inductive split, where we then used the function make_transductive, to make it transductive.

For CTD_DDA the same as for NDFRT_DDA were done, but while also taking the NDFRT_DDA train, test and validation set into consideration.

To then make the split for all of the datasets combined, it was simply to combine all of the train sets for the full train set, all of the test sets for the full test set and all of the validation sets for the full validation set.

Adding the new edges
The last two splits which were needed to be made were for the NDFRT_DDA dataset with the new edges, which we found in the data augmentation, Section 5.4 and the full split with these edges added. Since all of these edges were connected to a clinical drug and neither PharmKG8k nor CTD_DDA had any clinical drugs, and none of the new edges made, existed in any form in the NDFRT_DDA dataset, it was impossible to introduce data leakage. Therefore the only two things that we needed to take into account were to keep the splits transductive and at the same time, not move any edges which were already in the NDFRT_DDA train, test or validation sets, to keep the comparison between the splits fair.

The best way to do this was to split the new edges up into a train, test and validation set and first then, add them to respectively the NDFRT_DDA dataset, and the full dataset.

The way the new edges were split up, was by finding all of the edges which had entities that were not in NDFRT_DDA. There were only 26 of these entities, and these entities were in 61 edges. These edges were put into the train set, to ensure that the datasets splits which got the new edges also were transductive.

From there we found out that the rest of the entities in the new edges were a subset of all of the entities in the NDFRT_DDA train set. Therefore we could freely sample from these edges into the train, test and validation split. This kept the split transductive. Lastly, these edges were then added to the NDFRT_DDA dataset, and to the full dataset.

7 Experimental evaluation

In this section will we first try to reproduce PharmKG8k result show in their paper, this will be done in Section 7.1. Afterwards, in Section 7.2, will we present the result of different splits made on different datasets. The split is transductive with and without data leakage, and inductive without data leakage.

In Section 7.1 we will use the mean reciprocal rank, as a measure of the accuracy of the model, it ranges from 0 to 1, where 1 is a perfect score and 0 is the worst. We will also use hits@1, hits@3 and hits@10. The hits@1 signifies the percentage of cases where the true link, is in the top 1 of all possible links. hits@3 signifies the percentage of cases where the true link, is in the top 3 of all possible links and hits@10 signifies the percentage of cases where the true link, is in the top 10 of all possible links [40].

7.1 Reproducibility

To find out which models we would run, and to get the specific hyperparameters, we wanted to recreate the results of the models' of PharmKG8k. The reason we do not want to recreate BioNev's results is that their edges do not have relations, meaning that we would not be able to use the same models as the BioNev paper used on the combined datasets nor on PharmKG8k, without removing the relations in PharmKG8k. We also want to investigate how their split has affected their results.

PharmKG8k states the following: *All the baselines were trained for 100 to 1000 epochs using margin ranking loss with learning rate [0.005, 0.01, 0.1] and batch size [256, 512, 1024]. Other hyper-parameters for each approach were set at their default settings, as recommended by the Pykeen package [41]* [12]. With this information and the fact that they have results from 10 different models, in the worst-case scenario we have to run 81,000 different models; which is not feasible.

Therefore we have chosen the models that show the best results in the PharmKG8k, which are respectively ComplEx, ConvKB and TransE. In addition, we have also chosen to test exclusively with 100, 250, 500 and 1000 epochs. By doing so, we reduce the number of models we have to test to 108, note that this is still the worst-case scenario, if we find a result that matches, we use that model's hyper-parameters settings.

When running ComplEx with 1000 epochs, a learning rate of 0.005 and a batch size of 1024 we got an MRR of 0.1077 and the PharmKG8k paper got 0.107 and since they only included 3 decimals we are at most 0.012 off their result, but we can't be sure that this are the hyper-parameter that they have used.

When running TransE with 500 epochs, a learning rate of 0.005 and a batch size of 1024 we got an MRR of 0.0550, which is quite far from what they got in the PharmKG8K paper, which was 0.091. This is the closest value we got to what the PharmKG8k paper got. Since none of the results is close enough to the values shown in the PharmKG8k paper, we will not continue running models with these hyper-parameters based on our findings in this section. Instead, will we use the hyper-parameters used in the experiments section on the Pykeen GitHub. For instance, for the TransE model, we use the file *nguyen2018_transe_fb15k237.json* [42].

7.2 Experimental comparison of splits and different datasets

Due to time constraints, we were only able to do hyper-parameter optimizations, as a proof of concept for one of the model and dataset split combinations. All the numbers for Tables 8,9 and 10 are from models where no hyperparameter tuning was done. This means that the hyperparameter that we choose based on Pykeen templates, can have a big impact on what performs the best. These Pykeen templates hyperparameters might just perform better on one dataset or split, than another.

When comparing the PharmKG8k splits results, on the different models, the best-performing split among the PharmKG8k splits, is unsurprisingly the transductive split with data leakage. This is of course because it is a lot easier to predict edges which have already been seen, just with a different relation type. So this inflates the numbers since this is not what we actually want to test, since this is not a very interesting finding.

The next best split of PharmKG8k is the transductive split without data leakage. This makes also makes sense since the inductive split without data leakage actually is just downsampled, so this means that we have less data in the test and validation set. This will, of course, make it rarer to get a "correct" edge, since there are fewer of them.

When looking at the NDFRT_DDA dataset it gives a bit more of a mixed picture, if the new edges added to it actually give better results. For the ComplEx model, which can be seen in Table 8, it is no doubt better with the new edges added. Though in both ConvKB and TransE there are some metrics which are better for NDFRT_DDA without the new edges. Though there are still more metrics that signify that the NDFRT_DDA dataset with new edges is better.

For the merged dataset, the added edges give an incredible improvement when running the ComplEx model. Surprisingly the same can not be seen for the ConvKB model and the TransE model. Here the datasets perform almost the same, with the merged dataset without new edges performing slightly better.

When comparing all of the datasets with splits that are transductive without data leakage, the ComplEx model gives the best results for the NDFRT_DDA with new edges dataset, the ConvKB model gives the best results for the CTD_DDA dataset and TransE gives the best results for PharmKG8k. These are some surprising results since all of the models have different datasets which gives the best results, and the fact that usually more data yields better results, but none of the models gives the best results for the largest dataset, the merged dataset with the new edges.

Hyperparameter optimization of the ComplEx model on NDFRT_DDA

To see what kind of prediction accuracy we are missing out on, by not doing hyperparameter optimization, we have decided to do it on the NDFRT_DDA transductive split without the new edges for the ComplEx model. The hyperparameter optimization was done using a random search in a specified grid. It had 30 trials and was done on 5 different hyperparameters, where all of them had 3 possible values. This amounts to 243 different possible combinations of these hyperparameters in the grid. Since there were 30 trials, it means that 30 out of the 243 different combinations were tested. This optimization already took a couple of hours to run, so it quickly becomes apparent why it is mostly not suitable to do an exhaustive grid search, especially when you take into account the fact, that more possible values and more hyperparameters should be added if you want to find some extremely good hyperparameters.

The hyperparameters which were chosen for this model can be found in the appendix in Table 16. The MRR that this model found was 0.33009 which is about 30% better than the previous model without hyperparameter optimization. The hits at 1, 3 and 10 respectively were 0.20733, 0.37589 and 0.58879, which also all are significantly better.

Since this model performs a lot better after hyperparameter optimization, it would be expected that most of the other models also would perform better, but it is not known how much each model would improve. It can not be known if we would have seen vastly different results and reached other conclusions about the different splits if there were done hyperparameter optimizations for all of them.

8 Data privacy and security considerations: Reflections on our project

In this section, we will reflect on issues of data privacy and security related to our project. First of all, we will discuss whether the information we have is private information. All of the datasets we use are publically available, except the UMLS data used for data alignment in Section 5.2. This data requires an application to get access to but nevertheless is still quite easy to get access to. Therefore we conclude that some of the data we use is private, but at the same time, it does not have any major privacy concerns, as long as we do not redistribute the UMLS dataset.

But to investigate privacy concerns in relation to our

Table 7: Overview of our results that are closest to PharmKG8k's results, and comparison with the PharmKG8k results.

Models	[batch size, learning rate, epochs]	Results	MRR	N=1	N=3	N=10
ComplEx	-	PharmKG8k	0.107	0.046	0.110	0.225
	[1024, 0.005, 1000]	Our	0.108	0.047	0.110	0.225
ConvKB	-	PharmKG8k	0.106	0.052	0.107	0.209
	[512, 0.01, 500]	Our	0.100	0.042	0.103	0.212
TransE	-	PharmKG8k	0.091	0.034	0.092	0.198
	[1024, 0.005, 500]	Our	0.054	0.022	0.048	0.110

Table 8: The results of the ComplEx model, used on different types of splits. The underline signifies the best split for each type of dataset, that the spilt is made from.

	ComplEx				
Dataset	Split	MRR	hits@1	hits@3	hits@10
PharmKG8k	Transductive split without data leakage	0.01270	0.00449	0.00973	0.02323
	Transductive split with data leakage	0.02882	0.01334	0.02446	0.04937
	Inductive without data leakage	0.00870	0.00357	0.00665	0.01663
NDFRT_DDA	Transductive split	0.25331	0.13748	0.29010	0.49663
	New edges transductive split	0.29231	0.16669	0.33934	0.55230
CTD_DDA	Transductive split	0.02617	0.00684	0.01910	0.05192
Merged	Transductive split without data leakage	0.00029	0.00001	0.00005	0.00020
	New edges transductive split without data leakage	0.00571	0.00110	0.00377	0.01054

Table 9: The results of the ConvKB model, used on different types of splits. The underline signifies the best split for each type of dataset, that the spilt is made from.

	ConvKB				
Dataset	Split	MRR	hits@1	hits@3	hits@10
PharmKG8k	Transductive split without data leakage	0.00802	0.00319	0.00672	0.01532
	Transductive split with data leakage	0.02850	0.01349	0.02641	0.05097
	Inductive without data leakage	0.00713	0.00272	0.00569	0.01303
NDFRT_DDA	Transductive split	0.01060	0.00097	0.00345	0.01797
	New edges transductive split	0.01058	0.00110	0.00532	0.01856
CTD_DDA	Transductive split	0.01160	0.00157	0.00581	0.02311
Merged	Transductive split without data leakage	0.00618	0.00216	0.00470	0.01103
	New edges transductive split without data leakage	0.00618	0.00223	0.00459	0.01099

Table 10: The results of the TransE model, used on different types of splits. The underline signifies the best split for each type of dataset, that the spilt is made from.

	TransE				
Dataset	Split	MRR	hits@1	hits@3	hits@10
PharmKG8k	Transductive split without data leakage	0.03254	0.02545	0.03029	0.04237
	Transductive split with data leakage	0.04735	0.02549	0.04451	0.08454
	Inductive without data leakage	0.03190	0.02552	0.02969	0.04129
NDFRT_DDA	Transductive split	0.01113	0.00221	0.00646	0.02410
	New edges transductive split	0.00321	0.00775	0.02184	0.01097
CTD_DDA	Transductive split	0.01028	0.00233	0.00743	0.02072
Merged	Transductive split without data leakage	0.02466	0.01878	0.02284	0.03316
	New edges transductive split without data leakage	0.02433	0.01850	0.02258	0.03266

project, you could imagine a big firm such as Novo Nordisk, wanting to help researchers and share their data, but at the same time not wanting to share so much, such that their own research and drug development would be jeopardized. One thing they could choose to do is only share the data, from which their patents already are out of date. This would be data from 20 years ago [43]. This data would not have any privacy concerns, since the patents already are out of date.

Since data from 20 years ago, might not be very interesting to do research on, they might want to share some of their newer data. But they do not want others to be able to develop the same drugs, as they currently are working on. This then makes this new data sensitive, since it might hurt their business if they publish their data unaltered.

Then this raises these questions:

- Is it possible to alter the data in such a way, that it will not hurt their business, but still will be useful to researchers?
- If they choose to only release part of their data, how small of a subset of the data does it need to be if you want to avoid people being able to find out what some of the unreleased data is using link prediction?

Publishing the data without disclosing company secrets and making it useful to researchers would require finding the perfect subset, such that the essence of the data is still preserved and at the same time keeping the company's secrets.

If Novo Nordisk then chooses to only release a smaller subset of the data, then they can reduce their risk of disclosing sensitive information. But this alone does not protect them if people want to do link prediction to find some of the unreleased data. To minimize this, they can use techniques such as k-anonymity. K-anonymity describes how well the data is obscured. K signifies how many of each combination of quasi-identifiers that at least should exist, for every datapoint that exists. This means that each datapoints quasi-identifiers should be in at least $k - 1$ other data points. More specifically if a dataset has $k = 2$ it is said to be 2-anonymous. This means that each datapoints quasi-identifiers should be in at least 1 other datapoint [44]. The larger the value of k, the bigger the released subset of data can be, without taking too big of a risk, in regards to people using link prediction to link specific data points to the unreleased data.

9 Reflective analysis and conclusion

Reflection

It should be noted that most of the contents of this project were not something we learned in earlier courses. In the machine intelligence course, we scratched the surface of neural networks, but the embeddings of graphs and the specific embedding models are all new material to us. This means that a lot of time was spent learning these different concepts and getting familiar with the vocabulary.

One of our limitations of the link prediction problem is the lack of domain knowledge since we only have had a spare introduction to the domain of link prediction and neural networks. The overall learning curve was high, not only caused by a lack of domain knowledge but also due to the fact, the PharmKG8k dataset that we choose to work on was a wolf in sheep's clothing. We chose the PharmKG8k dataset with the expectation that everything was as it should be, with no data leakage and transductive. This was realised too late, so this forced us to change the direction of the project since it was needed to fix these problems with the PharmKG8k dataset before it was usable.

Retrospectively we should have checked for data leakage and unseen entities in the test and validation sets, but on the other hand, it was very unexpected.

Conclusion The problem of link prediction is a task which requires the right dataset, together with the right split. But even having that, does not always yield the best results. In our case, this could be due to that we have not used hyperparameter optimization. Nevertheless, we were able to quantify the effect of data leakage, which where clear when comparing the PharmKG8k splits. All the transductive splits with data leakage had the best accuracy in all statistics except for hits@1 in TransE. And since we do not want to predict these edges, since it is not very interesting relationships to learn, it gives us a false impression of the model's ability to do link prediction.

It was possible to merge PharmKG8k, CTD_DDA and NDFRT_DDA by using the UMLS_CUI. This made it possible to investigate if more data is equal to better results, this was done by comparing the different transductive splits without data leakage. It gives inconsistent results between the models, but none of the models has the merged dataset as the best performing. Therefore it does not seem to be true, that adding more data makes a better model, at least in our case.

It was possible to add new entities and edges to NDFRT_DDA to improve the connectivity to CTD_DDA and PharmKG8k. This was possible using an API to the UMLS database, to link the UMLS_CUI's to a broader concept. In most of the cases, the NDFRT_DDA with new edges and NDFRT_DDA had similar results, but in the ComplEx model, where we have the merged split with new edges, we see an extreme improvement. And therefore we can conclude that the data argumentation either had a positive effect or no effect.

The proof-of-concept hyperparameter optimization done on the NDFRT_DDA split gives by far the best results compared to all the results in Table 8, Table 9 and Table 10, this suggests having run hyperparameter optimization on all the splits, would give significantly better results. Since we have not done hyperparameter optimization on all of the different possible combinations of model, dataset and split we can not be sure that the conclusions we reach based on these are actually true, or if the conclusions reached would have been significantly different if we had done hyperparameter optimization.

10 Bibliography

[1] David Taylor. "The Pharmaceutical Industry and the Future of Drug Development". In: *Pharmaceuticals in the Environment.* The Royal Society of Chemistry, Sept. 2015. ISBN: 978-1-78262-189-8. DOI: 10.1039/9781782622345-00001. URL: https://doi.org/10.1039/9781782622345-00001.

[2] *The Drug Development Process.* [Online; accessed 4. Apr. 2023]. Apr. 2018. URL: https://www.fda.gov/patients/learn-about-drug-and-device-approvals/drug-development-process.

[3] Y. Cha et al. "Drug repurposing from the perspective of pharmaceutical companies". In: *Br. J. Pharmacol.* 175.2 (Jan. 2018), p. 168. DOI: 10.1111/bph.13798.

[4] Kevin McCoy et al. "Biomedical Text Link Prediction for Drug Discovery: A Case Study with COVID-19". In: *Pharmaceutics* 13.6 (2021). ISSN: 1999-4923. DOI: 10.3390/pharmaceutics13060794. URL: https://www.mdpi.com/1999-4923/13/6/794.

[5] Great Learning Team. "What is Cross Validation in Machine learning? Types of Cross Validation". In: *Great Learning Blog: Free Resources what Matters to shape your Career!* (June 2022). URL: https://www.mygreatlearning.com/blog/cross-validation.

[6] pykeen. *pykeen.* [Online; accessed 7. Mar. 2023]. Mar. 2023. URL: https://github.com/pykeen/pykeen.

[7] Xiang Yue et al. "Graph embedding on biomedical networks: methods, applications and evaluations". In: *Bioinformatics* 36.4 (2020), pp. 1241–1251.

[8] John Morrell. *Does More Data Equal Better Analytics?* [Online; accessed 25. May 2023]. July 2021. URL: https://www.datameer.com/blog/does-more-data-equal-better-analytics.

[9] *DrugBank | Drug Database.* [Online; accessed 25. May 2023]. May 2023. URL: https://www.drugbank.com/datasets.

[10] Meihong Wang, Linling Qiu, and Xiaoli Wang. "A Survey on Knowledge Graph Embeddings for Link Prediction". In: *Symmetry* 13.3 (Mar. 2021), p. 485. ISSN: 2073-8994. DOI: 10.3390/sym13030485.

[11] pykeen. *ilpc2022.* [Online; accessed 25. May 2023]. May 2023. URL: https://github.com/pykeen/ilpc2022.

[12] Shuangjia Zheng et al. "PharmKG: a dedicated knowledge graph benchmark for bomedical data mining". In: 22.4 (July 2021). ISSN: 1477-4054. DOI: 10.1093/bib/bbaa344.

[13] James Chen. *What are Neural Networks? | IBM.* [Online; accessed 23. May 2023]. May 2023. URL: https://www.ibm.com/topics/neural-networks.

[14] Enzo Grossi and Massimo Buscema. "Introduction to artificial neural networks". In: *Eur. J. Gastroenterol. Hepatol.* 19.12 (Jan. 2008), pp. 1046–54. ISSN: 0954-691X. DOI: 10.1097/MEG.0b013e3282f198a0.

[15] Team. "Main Types of Neural Networks and its Applications Tutorial". In: *Towards AI* (Mar. 2022). URL: https://towardsai.net/p/machine-learning/main-types-of-neural-networks-and-its-applications-tutorial-734480d7ec8e.

[16] Jason Brownlee. "Difference Between a Batch and an Epoch in a Neural Network - MachineLearningMastery.com". In: *MachineLearningMastery* (Aug. 2022). URL: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch.

[17] Daksh Trehan. "Gradient Descent Explained - Towards Data Science". In: *Medium* (Dec. 2021). URL: https://towardsdatascience.com/gradient-descent-explained-9b953fc0d2c.

[18] Will Koehrsen. "Neural Network Embeddings Explained - Towards Data Science". In: *Medium* (Oct. 2018). URL: https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526.

[19] Dhruvil Karani. "Introduction to Word Embedding and Word2Vec - Towards Data Science". In: *Medium* (Dec. 2022). ISSN: 6520-2060. URL: https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa.

[20] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space.* 2013. arXiv: 1301.3781 [cs.CL].

[21] Emerging Technology from the ArXiv. "King Man + Woman = Queen: The Marvelous Mathematics of Computational Linguistics". In: *MIT Technology Review* (Apr. 2020). URL: https://www.technologyreview.com/2015/09/17/166211/king-man-woman-queen-the-marvelous-mathematics-of-computational-linguistics.

[22] *Dimensionality of Word Embeddings | Baeldung on Computer Science.* [Online; accessed 19. May 2023]. Mar. 2023. URL: https://www.baeldung.com/cs/dimensionality-word-embeddings.

[23] Antoine Bordes et al. “Translating Embeddings for Modeling Multi-relational Data”. In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper_files/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf.

[24] Théo Trouillon et al. “Complex Embeddings for Simple Link Prediction”. In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 2071–2080. URL: https://proceedings.mlr.press/v48/trouillon16.html.

[25] Dai Quoc Nguyen et al. “A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 327–333. DOI: 10.18653/v1/N18-2053. URL: https://aclanthology.org/N18-2053.

[26] Tarang Shah. “About Train, Validation and Test Sets in Machine Learning”. In: *Medium* (May 2021). URL: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

[27] Prashant Gupta. “Regularization in Machine Learning - Towards Data Science”. In: *Medium* (June 2018). URL: https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a.

[28] Gianluca Malato. *Hyperparameter tuning. Grid search and random search.* [Online; accessed 23. May 2023]. May 2021. URL: https://www.yourdatateacher.com/2021/05/19/hyperparameter-tuning-grid-search-and-random-search.

[29] Michael Galkin. *Inductive Link Prediction in Knowledge Graphs - Towards Data Science.* Towards Data Science, Mar. 2022. URL: https://towardsdatascience.com/inductive-link-prediction-in-knowledge-graphs-23f249c31961.

[30] Jason Brownlee. “How to Avoid Data Leakage When Performing Data Preparation - MachineLearningMastery.com”. In: *MachineLearningMastery* (Aug. 2020). URL: https://machinelearningmastery.com/data-preparation-without-data-leakage.

[31] Prerna Singh. “Data Leakage in Machine Learning: How it can be detected and minimize the risk”. In: *Medium* (Jan. 2022). ISSN: 8439-7562. URL: https://towardsdatascience.com/data-leakage-in-machine-learning-how-it-can-be-detected-and-minimize-the-risk-8ef4e3a97562.

[32] Farahnaz Akrami et al. “Realistic Re-evaluation of Knowledge Graph Completion Methods: An Experimental Study”. In: *arXiv* (Mar. 2020). DOI: 10.48550/arXiv.2003.08001. eprint: 2003.08001.

[33] *The Comparative Toxicogenomics Database | CTD.* [Online; accessed 17. May 2023]. May 2023. URL: http://ctdbase.org.

[34] *National Drug File - Reference Terminology - Summary | NCBO BioPortal.* [Online; accessed 17. May 2023]. May 2023. URL: https://bioportal.bioontology.org/ontologies/NDFRT.

[35] biomed-A. I. *PharmKG.* [Online; accessed 8. May 2023]. May 2023. URL: https://github.com/biomed-AI/PharmKG.

[36] *Unified Medical Language System (UMLS).* [Online; accessed 22. Feb. 2023]. Feb. 2023. URL: https://www.nlm.nih.gov/research/umls/index.html.

[37] *UMLS Metathesaurus Browser.* [Online; accessed 5. Apr. 2023]. Mar. 2023. URL: https://uts.nlm.nih.gov/uts/umls/concept/C0245514.

[38] *UMLS Metathesaurus Browser.* [Online; accessed 5. Apr. 2023]. Mar. 2023. URL: https://uts.nlm.nih.gov/uts/umls/concept/C0693738.

[39] *UMLS API Home.* [Online; accessed 14. Apr. 2023]. Mar. 2023. URL: https://documentation.uts.nlm.nih.gov/rest/home.html.

[40] Rebecca Braken et al. “An Evaluation of Link Prediction Approaches in Few-Shot Scenarios”. In: *Electronics* 12.10 (May 2023), p. 2296. ISSN: 2079-9292. DOI: 10.3390/electronics12102296.

[41] Mehdi Ali et al. “BioKEEN: a library for learning and evaluating biological knowledge graph embeddings”. In: *Bioinformatics* 35.18 (Sept. 2019), pp. 3538–3540. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btz117. eprint: 30768158.

[42] pykeen. *pykeen.* [Online; accessed 18. May 2023]. May 2023. URL: https://github.com/pykeen/pykeen/blob/master/src/pykeen/experiments/transe/bordes2013_transe_fb15k.yaml.

[43] *Article 63 Term of the European patent.* [Online; accessed 23. May 2023]. May 2023. URL: https://new.epo.org/en/legal/epc/2016/a63.html.

[44] Heather Devane. “Everything You Need to Know About K-Anonymity”. In: *Immuta* (Apr. 2023). URL: https://www.immuta.com/blog/k-anonymity-everything-you-need-to-know-2021-guide.

11 Appendix

11.1 Tables with the semantic types of the different datasets

Table 11: Semantic types of NDFRT_DDA's nodes that only appear in NDFRT_DDA

Semantic type	NDFRT_DDA
"Clinical Drug"	64.10%
"Drug Delivery Device"	0.86%
"Amino Acid, Peptide, or Protein", "Pharmacologic Substance", "Immunologic Factor", "Indicator, Reagent, or Diagnostic Aid"	0.07%
Pharmacologic Substance", "Immunologic Factor", "Indicator, Reagent, or Diagnostic Aid"	0.07%
"Virus"	0.04%
"Bacterium"	0.03%
"Fungus"	0.03%
"Age Group"	0.01%
"Amino Acid, Peptide, or Protein", "Indicator, Reagent, or Diagnostic Aid"	0.01%
"Amino Acid, Peptide, or Protein", "Pharmacologic Substance", "Indicator, Reagent, or Diagnostic Aid"	0.01%
"Nucleic Acid, Nucleoside, or Nucleotide", "Vitamin"	0.01%
"Clinical Drug", "Drug Delivery Device"	0.01%
"Nucleic Acid, Nucleoside, or Nucleotide", "Pharmacologic Substance", "Vitamin"	0.01%
"Physiologic Function"	0.01%

Table 12: Semantic types of PharmKG8k's nodes that only appear in PharmKG8k

Semantic types	PharmKG8k
"Gene or Genome"	52.94%
"Amino Acid, Peptide, or Protein", "Receptor"	0.85%
"Molecular Function"	0.27%
"Amino Acid, Peptide, or Protein", "Immunologic Factor", "Receptor"	0.25%
"Body Part, Organ, or Organ Component"	0.22%
Amino Acid, Peptide, or Protein", "Enzyme", "Receptor"	0.20%
"Language"	0.13%
"Body Location or Region"	0.10%
"Intellectual Product"	0.06%
"Quantitative Concept"	0.06%
"Laboratory Procedure"	0.04%
"Qualitative Concept"	0.03%
"Geographic Area"	0.03%
"Fish"	0.01%
"Hormone", "Indicator, Reagent, or Diagnostic Aid"	0.01%
"Idea or Concept"	0.01%
"Professional Society"	0.01%
"Spatial Concept"	0.01%
"Cell"	0.01%
"Cell Function"	0.01%
"Amino Acid, Peptide, or Protein", "Enzyme", "Immunologic Factor"	0.01%
"Temporal Concept"	0.01%
"Activity"	0.01%
"Molecular Biology Research Technique"	0.01%
"Receptor"	0.01%
"Diagnostic Procedure"	0.01%
"Classification"	0.01%

Table 13: Semantic types of CTD_DDA's nodes that only appear in CTD_DDA

Semantic type	CTD_DDA
"Hazardous or Poisonous Substance", "Inorganic Chemical"	0.36%
"Biologically Active Substance"	0.34%
"Hazardous or Poisonous Substance"	0.20%
"Amino Acid, Peptide, or Protein", "Biologically Active Substance", "Hazardous or Poisonous Substance"	0.19%
"Organic Chemical", "Vitamin"	0.12%
"Chemical Viewed Structurally"	0.11%
"Indicator, Reagent, or Diagnostic Aid", "Inorganic Chemical"	0.11%
"Amino Acid, Peptide, or Protein", "Hazardous or Poisonous Substance"	0.10%
"Biologically Active Substance", "Hazardous or Poisonous Substance"	0.09%
"Organic Chemical", "Biologically Active Substance", "Hazardous or Poisonous Substance"	0.09%
"Organic Chemical", "Indicator, Reagent, or Diagnostic Aid", "Hazardous or Poisonous Substance"	0.09%
"Organic Chemical", "Hormone"	0.08%
"Chemical Viewed Functionally"	0.07%
"Indicator, Reagent, or Diagnostic Aid", "Hazardous or Poisonous Substance", "Inorganic Chemical"	0.04%
"Amino Acid, Peptide, or Protein", "Enzyme", "Hazardous or Poisonous Substance"	0.03%
"Amino Acid, Peptide, or Protein", "Immunologic Factor", "Hazardous or Poisonous Substance"	0.02%
"Organic Chemical", "Chemical Viewed Functionally"	0.02%
"Nucleic Acid, Nucleoside, or Nucleotide", "Indicator, Reagent, or Diagnostic Aid"	0.02%
"Biologically Active Substance", "Inorganic Chemical"	0.02%
"Hormone"	0.02%
"Chemical"	0.02%
"Indicator, Reagent, or Diagnostic Aid", "Element, Ion, or Isotope"	0.02%
"Pharmacologic Substance", "Hazardous or Poisonous Substance"	0.02%
"Amino Acid, Peptide, or Protein", "Pharmacologic Substance", "Vitamin"	0.02%
"Organic Chemical", "Biologically Active Substance", "Antibiotic"	0.01%
"Nucleic Acid, Nucleoside, or Nucleotide", "Biologically Active Substance", "Antibiotic"	0.01%
"Amino Acid, Peptide, or Protein", "Hormone", "Indicator, Reagent, or Diagnostic Aid"	0.01%
"Nucleic Acid, Nucleoside, or Nucleotide", "Amino Acid, Peptide, or Protein", "Pharmacologic Substance"	0.01%
"Chemical Viewed Structurally", "Biomedical or Dental Material"	0.01%
"Nucleic Acid, Nucleoside, or Nucleotide", "Amino Acid, Peptide, or Protein", "Enzyme"	0.01%
"Disease or Syndrome", "Anatomical Abnormality"	0.01%
"Cell Component"	0.01%
"Amino Acid, Peptide, or Protein", "Biologically Active Substance", "Indicator, Reagent, or Diagnostic Aid"	0.01%
"Amino Acid, Peptide, or Protein", "Vitamin"	0.01%
"Organic Chemical", "Immunologic Factor", "Hazardous or Poisonous Substance"	0.01%
"Hazardous or Poisonous Substance", "Element, Ion, or Isotope", "Inorganic Chemical"	0.01%
"Natural Phenomenon or Process"	0.01%
"Nucleic Acid, Nucleoside, or Nucleotide", "Hazardous or Poisonous Substance"	0.01%
"Nucleic Acid, Nucleoside, or Nucleotide", "Amino Acid, Peptide, or Protein"	0.01%
"Organic Chemical", "Pharmacologic Substance", "Element, Ion, or Isotope"	0.01%
"Environmental Effect of Humans"	0.01%
"Pharmacologic Substance", "Hormone", "Vitamin"	0.01%
"Vitamin"	0.01%
"Organic Chemical", "Element, Ion, or Isotope"	0.01%

Table 14: Semantic types of all the datasets nodes that only appear in two of the datasets

Semantic type	NDFRT_DDA	CTD_DDA	PharmKG8k
"Substance"	0.16%	0.07%	0.00%
"Biomedical or Dental Material"	0.16%	0.09%	0.00%
"Organic Chemical", "Pharmacologic Substance", "Food"	0.05%	0.05%	0.00%
"Pharmacologic Substance", "Indicator, Reagent, or Diagnostic Aid", "Inorganic Chemical"	0.04%	0.08%	0.00%
"Biologically Active Substance", "Element, Ion, or Isotope"	0.04%	0.09%	0.00%
"Inorganic Chemical"	0.04%	0.71%	0.00%
"Organic Chemical", "Biomedical or Dental Material"	0.04%	0.20%	0.00%
"Organic Chemical", "Food"	0.03%	0.05%	0.00%
"Body Substance"	0.02%	0.03%	0.00%
"Antibiotic"	0.02%	0.04%	0.00%
"Element, Ion, or Isotope"	0.01%	0.21%	0.00%
"Experimental Model of Disease", "Neoplastic Process"	0.01%	0.09%	0.00%
"Organism Attribute"	0.01%	0.05%	0.00%
"Hazardous or Poisonous Substance", "Element, Ion, or Isotope"	0.01%	0.09%	0.00%
"Mental Process"	0.01%	0.01%	0.00%
"Pharmacologic Substance", "Immunologic Factor", "Hazardous or Poisonous Substance"	0.01%	0.01%	0.00%
"Individual Behavior"	0.01%	0.01%	0.00%
"Pharmacologic Substance", "Food"	0.01%	0.01%	0.00%
"Cell or Molecular Dysfunction"	0.01%	0.15%	0.00%
"Amino Acid, Peptide, or Protein", "Immunologic Factor", "Indicator, Reagent, or Diagnostic Aid"	0.01%	0.01%	0.00%
"Amino Acid, Peptide, or Protein", "Pharmacologic Substance", "Hazardous or Poisonous Substance"	0.01%	0.04%	0.00%
"Finding", "Disease or Syndrome"	0.01%	0.01%	0.00%
"Pharmacologic Substance", "Hazardous or Poisonous Substance", "Element, Ion, or Isotope"	0.01%	0.01%	0.00%
"Pharmacologic Substance", "Biologically Active Substance", "Inorganic Chemical"	0.01%	0.02%	0.00%
"Amino Acid, Peptide, or Protein", "Enzyme"	0.00%	0.07%	2.86%
"Therapeutic or Preventive Procedure"	0.00%	1.61%	0.24%
"Amino Acid, Peptide, or Protein", "Hormone"	0.00%	0.02%	0.04%
"Congenital Abnormality", "Disease or Syndrome"	0.00%	0.04%	0.04%
"Nucleic Acid, Nucleoside, or Nucleotide"	0.00%	0.43%	0.03%
"Clinical Attribute"	0.00%	0.02%	0.03%
"Experimental Model of Disease"	0.00%	0.07%	0.03%
"Laboratory or Test Result"	0.00%	0.01%	0.01%
"Phenomenon or Process"	0.00%	0.01%	0.01%
"Organic Chemical", "Immunologic Factor"	0.00%	0.05%	0.01%
"Pharmacologic Substance", "Vitamin"	0.00%	0.02%	0.01%
"Genetic Function"	0.00%	0.01%	0.01%
"Pharmacologic Substance", "Indicator, Reagent, or Diagnostic Aid", "Element, Ion, or Isotope"	0.01%	0.00%	0.01%

Table 15: Semantic types of all the datasets nodes that appear in all of the datasets

Semantic type	NDFRT_DDA	CTD_DD	PharmKG8k
"Organic Chemical", "Pharmacologic Substance"	14.17%	33.57%	14.97%
"Disease or Syndrome"	5.90%	15.11%	9.64%
"Pharmacologic Substance"	1.97%	2.00%	0.06%
"Organic Chemical", "Antibiotic"	1.49%	1.89%	1.88%
"Neoplastic Process"	1.01%	2.88%	1.37%
"Amino Acid, Peptide, or Protein", "Pharmacologic Substance", "Immunologic Factor"	1.00%	0.26%	0.08%
"Amino Acid, Peptide, or Protein", "Pharmacologic Substance"	0.72%	1.60%	0.53%
"Pharmacologic Substance", "Inorganic Chemical"	0.63%	0.75%	0.15%
"Organic Chemical", "Pharmacologic Substance", "Hormone"	0.57%	0.82%	0.68%
"Amino Acid, Peptide, or Protein", "Pharmacologic Substance", "Hormone"	0.56%	0.31%	0.22%
"Pathologic Function"	0.55%	1.59%	0.92%
"Pharmacologic Substance", "Immunologic Factor"	0.52%	0.13%	0.01%
"Sign or Symptom"	0.44%	0.91%	0.40%
"Organic Chemical", "Pharmacologic Substance", "Vitamin"	0.43%	0.45%	0.35%
"Nucleic Acid, Nucleoside, or Nucleotide", "Pharmacologic Substance"	0.41%	0.91%	0.53%
"Mental or Behavioral Dysfunction"	0.31%	1.08%	0.47%
"Organic Chemical", "Pharmacologic Substance", "Biologically Active Substance"	0.27%	0.74%	0.15%
"Injury or Poisoning"	0.27%	0.85%	0.24%
"Medical Device"	0.27%	0.01%	0.03%
"Amino Acid, Peptide, or Protein", "Pharmacologic Substance", "Enzyme"	0.25%	0.04%	0.03%
"Finding"	0.21%	0.78%	0.46%
"Amino Acid, Peptide, or Protein", "Pharmacologic Substance", "Biologically Active Substance"	0.21%	0.16%	0.42%
"Organic Chemical"	0.18%	14.63%	0.21%
"Amino Acid, Peptide, or Protein", "Antibiotic"	0.17%	0.21%	0.20%
"Food"	0.17%	0.12%	0.01%
"Organic Chemical", "Pharmacologic Substance", "Hazardous or Poisonous Substance"	0.16%	0.34%	0.24%
"Organic Chemical", "Pharmacologic Substance", "Immunologic Factor"	0.11%	0.06%	0.03%
"Congenital Abnormality"	0.08%	0.81%	0.84%
"Organic Chemical", "Pharmacologic Substance", "Indicator, Reagent, or Diagnostic Aid"	0.07%	0.27%	0.10%
"Organic Chemical", "Hazardous or Poisonous Substance"	0.04%	3.48%	0.08%
"Pharmacologic Substance", "Biologically Active Substance", "Element, Ion, or Isotope"	0.04%	0.07%	0.01%
"Pharmacologic Substance", "Element, Ion, or Isotope"	0.04%	0.09%	0.04%
"Amino Acid, Peptide, or Protein", "Biologically Active Substance"	0.04%	0.73%	4.03%
"Organic Chemical", "Indicator, Reagent, or Diagnostic Aid"	0.03%	1.17%	0.21%
"Pharmacologic Substance", "Biologically Active Substance"	0.03%	0.34%	0.03%
"Nucleic Acid, Nucleoside, or Nucleotide", "Pharmacologic Substance", "Biologically Active Substance"	0.02%	0.05%	0.04%
"Pharmacologic Substance", "Indicator, Reagent, or Diagnostic Aid"	0.02%	0.02%	0.01%
"Organic Chemical", "Biologically Active Substance"	0.02%	1.72%	0.13%
"Organism Function"	0.01%	0.01%	0.01%
"Manufactured Object"	0.01%	0.04%	0.04%
"Acquired Abnormality"	0.01%	0.09%	0.06%
"Immunologic Factor"	0.01%	0.09%	0.03%
"Amino Acid, Peptide, or Protein", "Immunologic Factor"	0.01%	0.11%	1.06%
"Pharmacologic Substance", "Hormone"	0.01%	0.08%	0.01%
"Anatomical Abnormality"	0.01%	0.26%	0.08%
"Nucleic Acid, Nucleoside, or Nucleotide", "Biologically Active Substance"	0.01%	0.15%	0.03%
"Amino Acid, Peptide, or Protein"	0.01%	0.83%	0.06%
"Nucleic Acid, Nucleoside, or Nucleotide", "Antibiotic"	0.01%	0.02%	0.01%
"Organic Chemical", "Biologically Active Substance", "Indicator, Reagent, or Diagnostic Aid"	0.01%	0.01%	0.01%
"Organ or Tissue Function"	0.01%	0.03%	0.03%
"Pharmacologic Substance", "Hazardous or Poisonous Substance", "Inorganic Chemical"	0.01%	0.06%	0.01%
"Indicator, Reagent, or Diagnostic Aid"	0.01%	0.04%	0.01%

11.2 Hyperparameter optimization

Table 16: Hyperparameter optimization for the NDFRT_DDA dataset and the ComplEx model

Hyperparameter	Chosen value	Possible values
model.embedding_dim	100	[50, 100, 200]
optimizer.lr	0.001	[0.0001, 0.001, 0.01]
negative_sampler.num_negs_per_pos	100	[10, 50, 100]
training.num_epochs	150	[50, 100, 150]
training.batch_size	64	[32, 64, 128]