Knowledge Graph Embeddings to Implement Explainable AI

Ronan Murphy

In Partial Fulfilment of the Requirements for the Degree of Masters of Science in Computer Science

(Artificial Intelligence)

College of Engineering & Informatics

National University of Ireland, Galway

September 2020

Supervisor: Dr Michael Schukat

Acknowledgements

Firstly, I would like to sincerely thank my mentor Dr Michael Schukat for his excellent advice and support throughout this project. I wish to pay special regards also to Dr Maciej Dabrowski and Dr Emir Munoz of Genesys Galway of whom assisted me without hesitation, providing a broad range of knowledge and insight into how to approach and develop the research topic. Finally, I would like to thank my family and friends for their optimism, consultation and encouragement during this project.

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Masters in Science is entirely my own work, that I have exercised reasonable care to ensure the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Ronan Murphy Date: September 2020

Glossary

AI	Artificial Intelligence
KG	Knowledge Graph
ML	Machine Learning
KGE	Knowledge Graph Embedding
KGC	Knowledge Graph Completion
SRL	Statistical Relational Learning
SOTA	State-of-the-Art
DL	Deep Learning
XAI	Explainable AI
MLN	Markov Logic Networks
NLP	Natural Language Processing
SRM	Statistical Relational Models
BRN	Bayesian Relational Networks
RNN	Recurrent Neural Network
RDF	Resource Description Framework
YAGO	Yet Another Great Ontology
PITF	Pairwise Interaction Tensor Factorization
CD	Canonical Decomposition
AUC	Area Under the Curve
AMIE	Association Rule Mining under Incomplete Evidence
PRA	Path Ranking Algorithm
RWR	Random Walk with Restart
HolE	Holographic Embedding
ComplEx	Complex Embedding
MRR	Mean Reciprocal Rank
ER	Entity Resolution
PSL	Probabilistic Soft Logic
FEVER	Framework for Evaluating Entity Resolution
FEBRL	Freely Extensible Biomedical Record Linkage
MARLIN	Multiply Adaptive Record Linkage with Induction
MAP	‘Maximum A Posteriori’

CNN	Convolutional Neural Network
BRNN	Bidirectional Recurrent Neural Network
LSTM	Long Short Term Memory
CORD-19	COVID-19 Open Research Dataset
CSV	Comma Separated Values
AWS	Amazon Web Services
CKG	Covid-19 Knowledge Graph
LDA	Latent Dirichlet allocation
NLL	Negative Log-Likelihood
BCE	Binary-cross entropy
SGD	Stochastic Gradient Descent
LCWA	Local Closed World Assumption
PCA	Principal Component Analysis
SVD	Singular Value Decomposition
TSV	Tab-Separated Values
UMAP	Uniform Manifold Approximation and Projection for Dimension Reduction
T-SNE	T-distributed Stochastic Neighbour Embedding

Abstract

Today, Artificial Intelligence (AI) is at the forefront of modern technology as it provides applications of learning and self-improvement, which before were not possible. We are on the precipice of an exponential revolution comparable to that of the agricultural and industrial revolution, which will transform modern society as we know it. This transformation is an intelligence revolution of machines to reach Super-Intelligence, which Nick Bostrom defines as "any intellect that greatly exceeds the cognitive performance of humans in virtually all domains of interest" [1]. When this happens is unknown, as it depends on technology advancements, but the change is inevitable. Although this seems enticing, there are many precautions which we must take to avoid the potential adverse effects of these powerful machines. Taking this into consideration, 'Trust' is a major concern especially for hazardous tasks, such as autonomous surgery or driving. This project aims to provide more detailed explanations to the users about the decisions these agents make.

Knowledge graphs (KG) are models that represent the relationship between nodes in the graph called Entities. In recent times, these graphs have reached impressive sizes with billions of links used for storing and retrieving information. Google uses its ‘Knowledge Graph’, one of the world's most powerful [2], which is where the term originally derived. The ‘Knowledge Graph’ is used for Google’s search engine and is an excellent example of how a KG can enhance performance. One can find other examples of KG applications at all the biggest multinational technology companies - Microsoft's Satori, Facebook's Entities Graph and Amazon's E-commerce platform. As Machine Learning (ML) and KG progress, they do so in different directions, ML becoming more accurate at predicting specific tasks and KG improves its ability to represent entities and relational links very reliably, with a high level of Explainability to the user. Combining these tools have had tremendous results; modern applications include Netflix's Recommender and Amazon's Related-Items. These tools provide the user with more relevant feedback which they can customise to their specific needs. Using a KG has the potential to conquer problems in ML as it adds the ability to gain a deeper understanding of its process.

The objective of this project is to test the performance of knowledge graph embedded (KGE) models, trained on the same dataset. The output will predict classifications, but with the enhancement, the KG can explain more about the decisions of these classifications, giving a more in-depth examination of the semantic information. Different approaches will be tested and evaluated in the hope of creating an optimised system to improve the Explainability of an algorithm's selection for the user's benefit.

Table of Contents

Table of Figures

List of Tables

1 Introduction

1.1 Research Topic and Motivation

Artificial Intelligence is the learning of intelligent machines, to achieve and maximise the machine's defined goals. The research topic chosen for this project concerns the task of improving the Explainability of an algorithm by using a KG. Explainability, also known as interpretability, are techniques used in AI where the results, and the process of getting the results, can be understood by a human expert [3]. In this project, a dataset is trained on a knowledge graph embedded model to provide more interpretability to the user. The model will be able to provide both high accuracy in predicting classification and interpretability to the user about the model's prediction. These tasks create the separation of the project into the following sections:

1) Training different knowledge graph embedded models over a benchmark dataset.
2) Use the knowledge graph embeddings in the link prediction task for completion. The purpose of this is to find pairs of nodes which may form a link based on probability, to remove false positive links and to identify and connect false-negative predictions.
3) Once the graph is fully defined, mutual neighbours can be used to give more interpretability to the user about the predictions given.

1.2 Background

1.2.1 Trust in AI

The human trust of autonomous agents is a significant concern for the field of AI. One reason for the current lack of trust is due to the unexplainable decisions which these devices make for a classification task. Trust is critical for the progress of modern AI, mainly when applied to hazardous tasks, such as self- driving cars. If the decisions which these algorithms make were explainable to the user, with a detailed description, they could anticipate the output of tasks as expected behaviour. This project aims to provide interpretability for the decisions that a knowledge graph embedding (KGE) makes, which will not solve the issue entirely but will hopefully reduce the impact of the problem on developing AI. This problem is called Explainability, whereby the results of an AI task are explained to a user and can be easily understood.

There are many ways in which some form of Explainability can be achieved, all from a predefined dataset. The dataset must be large to enable accurate predictions of the features, representing a variety of linked information on a topic. Certain biases may occur if the prediction of a particular node occurs a disproportionate amount of times, misinterpreting the actual data. Retrieving the dataset from several different credible sources is an approach to reduce this effect. The goal of representing the dataset in a linked manner is to allow quick and precise retrieval. To this effect, models can generate a lower dimensionality of a large dataset which can have many applications such as link prediction and interpretability for the user. A knowledge graph embedding is one such model, which embeds the entities and relationships of a KG into low-dimensional vector spaces [4]. There are a couple of methods, similar to knowledge graphs embeddings, which can represent datasets effectively and efficiently for prediction tasks. These will be discussed in detail in the literature review section 2.3 to help determine the best approach.

1.2.2 Knowledge Graph Embedding

Knowledge graph embedding models are algorithms that generate vector representations of entities and relations in a lower dimensionality, from a KG. KGE are commonly used in the technology industry to represent large datasets which are linked based on semantically relevant similarities. KGE models produce a better representation of the dataset in vector format, which can be used for many prediction tasks "while preserving the inherent structure of the knowledge graph" [5]. By doing this, they can have many applications for KGs, including Link Prediction for knowledge graph completion (KGC), and relation extraction for Explainability. KGE is a form of Statistical Relational Learning (SRL) [6] as opposed to the standard feature learning we see in traditional ML. In SRL "the representation of an object can contain its relationships to other objects" [6], this allows a KG to learn the relations between entities intuitively.

There are many different approaches to create a KGE, but there are two preeminent categories, namely Latent Feature models, and Graph Feature models. Latent Feature models use features which are indirectly observable in the data to create relations between entities. These models have the task of inferring the characteristics from the dataset, derived from the interactions of the latent features of entities. Graph Feature models use observed edges in the KG to extract relevant features. Both of these methods have benefits, and therefore combining them can produce significant results. Latent feature models global relations efficiently, whereas Graph feature models work better with local patterns. This combinational approach has improved predictive performance and increased performance speed. Section 2.4 discusses different state-of-the-art (SOTA) techniques for these categories.

1.2.3 Link Prediction

Knowledge graphs have many applications, including question answering, recommender systems, semantic searching, and word disambiguation. These functions can be beneficial for a variety of research but need a completed KG to perform effectively. In recent years, KGs have become very large; academic testing examples include YAGO and Freebase, commercial products have KGs such as Google's Knowledge Graph and Microsoft's Satori. Although these KGs represent billions of relationships, they all include many missing and incorrect links, and therefore can be improved. Knowledge graph completion aims to enhance missing information in an incomplete knowledge graph automatically. KGC is also known as the task of Link Prediction for missing links and has many difficulties in trying to complete a network. The prediction task has many variables examples include, entities having a varying number of attributes, several types of relations in a KG, and the transformation of growing KGs overtime.

KGEs use Link Prediction for completion of the KG, defined as adding missing edges between entities which are not currently present. It is a fundamental technique to expand the relations in a KG leading to more accuracy and a broader network. Nickel et al. [6] prove that significant improvement in accuracy occurs when using Link Prediction in KGEs. It can be accomplished by a simple ranking to understand which links are most likely to relate. Implementation of Link prediction can be performed after the creation of the initial knowledge graph, to improve the number of links and the precision of these links significantly. There are many techniques to return the probabilities of connections between nodes in a graph, explained in section 2.5. When the KG has improved with Link Prediction, it will be more accurately be able to classify input data and give a comprehensive report on the likelihood of links between certain entities, even if they are not directly observable. KGC is a critical stage in the project, which will significantly improve the overall results of knowledge graph embedding.

1.2.4 Classification and Interpretability

A fusion between different sources to make the KG, transformed into the KGE, can help with the trustworthiness of the nodes for false positives and the correct relations between them.

A large knowledge graph creates a KGE through training, which then can make predictions from a set of input data. The performance of the KGE's predictions will determine the effectiveness of this approach. The project requires a high level of accuracy before the Explainability phase can begin for the forecasting with the knowledge graph embedding.

Once the KGE makes predictions, the goal is to explain the reasoning behind them, to give interpretably to the user. It is a difficult task to measure the Explainability of a system, especially for KGE as they represent elements in a vector space, which causes a reduction in intuitive interpretability. It is crucial to reduce biases in knowledge graph embeddings caused by dataset, which does not represent the information fairly. A decrease in bias can occur through the use of a large knowledge graph and Explainability for inferences which promote an unfair tendency for predictions. This project aims to provide more information to the user, than the predictions alone provide, to justify the behaviour of the system.

1.3 Aim of Research

The Explainable AI (XAI) problem entails the issue of trusting the decisions of autonomous devices. This project aims to provide a potential improvement of this problem in tasks in the ML landscape. The different sections will accomplish this.

1) Creating a knowledge graph, in the form of triples, linking between data nodes in a knowledge base (Section 3)
2) Apply knowledge graph embedding models to obtain vector representations of entities and relations (Section 4)
3) Link prediction methods are performed for KG completion, choosing the best performing - highest accuracy and broadest variety (Section 5)
4) Using this knowledge graph embedded model to create predictions and interpretability of these predictions (Section 6)

1.4 Structure of the Thesis

There are eight sections in this thesis. This section provides some background information and an overview of the project domain. Section 2 contains a detailed literature review discussing the current knowledge in the field by leading researchers. Chapters 3 – 6 specify the approaches and methodologies taken while undergoing this project. In the 7th section, the results obtained from the testing of different methods are displayed, followed by a detailed analysis. Finally, section 8 is the conclusion of the thesis, which provides an overall discussion of the results concerning the research topic and the future work to be completed.

2 Literature Review

2.1 Overview

Chapter 2 develops the reasoning behind this project, discussing the background research which other academics in the field of large semantic networks have made, and reviewing different approaches with comparisons to their results. The chapter contains five other subsections. Section 2.2 defines the importance of trust and explainability artificial intelligent systems, which leads to the purpose of this research. Section 2.3 details approaches to storing and gaining information from semantic categorical datasets. The primary approach to interpreting large semantic networks is through the use of knowledge graph embeddings. Section 2.4 provides detailed descriptions of specific research papers and SOTA models. Knowledge graph completion is the title of section 2.5, which is a method to improve the quality of a KG and can be completed through several methods discussed. The final section 2.6, examines the most modern techniques and applications of knowledge graphs in real-world situations.

2.2 Trust and Explainability

Keng Siau and Weiyu Wang discuss the topic of trust in AI, stating, "Trust is key in ensuring the acceptance and continuing progress and development of artificial intelligence." [7] This formulates the importance of developing Artificial Intelligent systems which the end-user can trust and interpret. An essential feature of 'Trust' is an algorithm's transparency which it should "be able to explain/justify its behaviours and decisions." [7]

A solution to this problem of trust is through the creation models which have an explainable decision process. As stated by Derek Doran et al., "explanations should provide insight into the rationale the AI uses to draw a conclusion." [8], when discussing the vitality of explaining the results of a model to a user. This paper suggests models which are comprehendible or interpretable enhancing the generic opaque black-box systems seen in ML. Ambiguity occurs when the user has very little knowledge of the system's internal workings, only the inputs and outputs. The paper notes that although models provide results, they do not give clarity to the user to comprehend its internal decisions. They state the need to "develop methods which might enable explicit automated reasoning over model properties and decision factors by extracting symbolic rules from connectionist models." [8] A connectionist model is an algorithmic system, such as a neural network, which derive from biological interaction between a human brain's neurons and the environment the person perceives. An integrated approach which provides the user with feedback about the decision process which can easily understand the network of information. The next section details the connectionist model types, which can predict and explain results.

2.3 Knowledge-Based Methods

2.3.1 Knowledge-Graph Embeddings

James Allen and Alan Frisch describe a semantic network as "the formalisation of knowledge retrievers as well as the representation languages that they operate on." [9] Through representing data in a tree network format, links are created between similar objects making comparisons and retrieval much more manageable. It enables "Semantic networks not only represent information but facilitate the retrieval of relevant facts" [9]. KGs link semantic objects developed similar to the World Wide Web and are a significant reason web browsers and leading social media sites use KGs to connect related documents for improved usability.

A knowledge graph is a type of semantic network with more constraints, which has a structure and attributes that are not all know when created. KG theory began in 1982 to represent knowledge for expert systems but later developed into the modern Natural Language Processing application. As discussed by Sri Nurdiati and Cornelis Hoede in the 25-year roadmap of KGs [10], they describe a knowledge-based system as one that combines knowledge from different places for a representation of natural language. This theory has developed to a sophisticated level where semantic web browsers such as Google have now implemented their own 'Knowledge Graph' in their search engine [2]. Google shows the benefits since adding their 'Knowledge Graph' to facilitate the browser reporting reduced ambiguity of a search, more detailed summaries and better overall performance, answering one-third of their users 100 million inquiries per month. A KG implemented to enhance an algorithm's performance, could see similar benefits.

KGs can appear across a wide variety of applications, and Lisa Ehrlinger and Wolfram Wöß attempt to define the concept in their paper. "A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge" [11], is their definition of KG. However, they state that labelling primitive ontologies as KGs is incorrect and an ontology network should only be defined as a KG if it follows preemptive measures. In essence, a KG performs reasoning and is not solely a knowledge base. The applications of KGs are vast, especially to their potential to solve the Explainability problem in AI.

2.3.2 Markov Logic Networks

Markov Logic Network (MLN) is a "first order knowledge base with a weight attached to each formula" [12], which can also achieve interpretability on a model by assigning weights to each entity in a knowledge-base. They are an alternative method to KGs, but both are Statistical Relational Models (SRM). MLNs combine first-order logic and probabilistic graphical models in a single representation and have many applications, including information extraction and Natural Language Processing (NLP). The standard MLN network is implemented as a greedy top-down algorithm, meaning it always chooses the best action possible when predicting. These greedy algorithms operate quickly but can lead to local maxima as the algorithm is exploitative. Mihalkova and Mooney [13] describe an alternative the greedy algorithm, detailing a bottom-up approach which avoids the exploitative issue. This approach effectively improves the accuracy over the standard top-down implementation. MLN still has disadvantages as it can only make decisions about variables in the model, and the algorithm has no understanding of the structure of the Markov Standard field – the overall network. For these reasons, KGEs can find more optimal solutions and give more accurate predictions for nodes.

2.3.3 Bayesian Relational Networks

Bayesian Relational Networks (BRN) instead of probabilistic rules, which is common in knowledge-based representation, use "probability distribution over the relations is directly represented by a Bayesian network" as described by Manfred Jaeger [14]. It provides the capabilities with probability distribution to represent relations as a network. This network produces a more semantically transparency and is an improvement over rule-based formalisms. Sutskever et al. describe a more modern approach using Bayesian Clustered Tensor Factorisation [15], which scales much better on large datasets, with high predictive accuracy and minimal per parameter selection. Bayesian methods have limitations as they are computationally expensive, and if cyclic relationships exist between nodes results suffer dramatically. Therefore, the rest of this review will focus on KGE implementations to try to provide an improvement for the Explainability problem.

2.4 Knowledge Graph Embedding Approaches

Nickel et al. explore how statistical models can use KGs for the completion of links [6]. They discuss this based on two types of statistical models, namely, Latent Feature Models which determine rules which are not directly observable, and Graph Feature models mining which map visible patterns in the graph. Resource Description Framework (RDF) is the format which each node is stored, mapping standard binary relationships in the form "(subject, predicate, and object)" [6], where subject and object are entity nodes, and the predicate is the relational link between them. The paper mentions multiple approaches for Latent Feature Models, including, RESCAL which examines triples based on pairwise comparisons, Matrix Factorisation methods where rows are subject/object and columns are predicate, Multi-Layer Perceptron, Neural Tensor Models, and Distance Models such as the 'TransE' [16]. Graph Feature Models extract features from observed edges in the graph used with Similarity Measures for Unirelational Data [17], Rule Mining [18], and Path Ranking algorithms [19]. Both approaches have their benefits while neither reigns superior. Therefore, a combination of their strengths can create the best model. The paper concludes, stating that although KGs are very powerful and have many uses, they miss out of common sense facts to humans which machines are not currently capable.

2.4.1 Latent Feature Models

The RESCAL algorithm is a bilinear model which Nickel and Tresp et al. [20] examine. A bilinear model has two or more independently linear variables which provide better structure than standard linear models. [21] This is an efficient algorithm which can compute factorisation, which is much faster than other state-of-the-art models. It shows that tensor-based approaches work effectively with relational learning. Tensor decompositions are beneficial as they are high-dimensional and sparse which give formidable results, although the sparsity can sometimes be an issue. Figure 1 displays the RESCAL factorisation between the two entities and their relation.

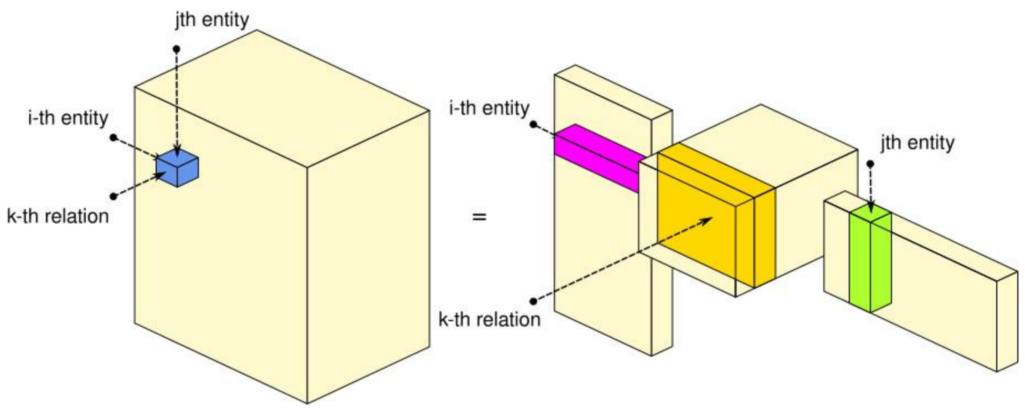


Figure 1 RESCAL factorisation [20]

Nickel, Krompaß et al. show an improvement when accounting for non-negativity [22]. Removing negative latent factors implements non-negativity and improves data sparsity by changing the update rules of multiplicative nodes. The results show an increase in the sparsity of the output with only a slight reduction in predictive accuracy and a slower convergence.

Riedel et al. [23] discuss Matrix Factorisation for relational extraction, which can learn latent feature vectors for entity tuples and relations and can outperform traditional methods, in terms of accuracy. This method provides the capabilities to run simultaneously on associations observed and allows reasoning about unstructured and structured data in mutually-beneficial ways. The knowledge graph structure is in the form of rows and columns, entity pairs as rows and relations as columns. YAGO [24] is an automatically extracted dataset from information sources, such as Wikipedia. The matrix factorisation approach compares to Nickel et al. to factorise YAGO for predicting links [25]. RESCAL tensor factorisation trains the YAGO dataset, as described above, for link prediction, which allows efficient predictions after tensor factorisation has occurred. Riedel's paper [23] states that the matrix method uses the ranking objective to complete the matrix, whereas the tensor factorisation method tries to learn the latent word embeddings. This approach is disadvantageous for matrix factorisation as learning these latent features is more computationally inefficient. Jenatton et al. [26] propose a method for modelling large multi-relational datasets. The model uses a bilinear structure to discover relations giving sparse latent factors. This model formulates the problem as a Matrix Factorisation and uses it to tackle real-world problems.

To predict the actual values of Resource Description Framework (RDF) triples of the graph, Drumond et al. [27] propose a method to which uses a 3D tensor representation of the RDF knowledge base. The feature applies tensor factorisation techniques which include the semantic information in predicting new triples from previously created ones. The tensor representation approach retrieves RDF datasets which use sparse tensors to complete the KG. Drumond's paper [27] proves that adding additionally relevant RDF relations the improves of the dataset. However, it states that results are not significantly increased based on the high computation of the Pairwise Interaction Tensor Factorization (PITF) model used. Sparse Canonical decomposition (CD) is another approach which has worse results but less runtime and can be preferable in some instances. Rendle and Schmidt-Thieme [28] dive further into the details of the PITF Tucker Decomposition model for its use in personalised tagging. They demonstrate the performance and usefulness of this linear runtime, which is more efficient for large-scale datasets compared to other standard methods.

Rendle discusses the scalability of these factorisation machines [29] for solving the issue by making use of repeating patterns in the design matrix of the relational structure of the data. This technique has applications for machine learning algorithms which are slow to process the large-scale of a knowledge base, by using coordinate descent to improve the algorithms. Implementing these changes enhances the speed and predictive Area Under the Curve (AUC) accuracy of the machine learning algorithms.

Miettinen [30] implements a Boolean Tensor Factorisation for binary multi-way problems through a more practical computation of the factorisation. The method maintains good sparsity and interpretability of the factors to an expert human. The results showed high accuracy and efficiency, but the ranking proved difficult, especially for real-valued problems.

Wang, Quan, et al. [31] discuss the approaches and applications of KGEs. It details specific techniques used in creating the KGE and then examines the applications in two categories In-KG and Out-of-KG. In-KG explains the applications of Link Prediction, Triple Truth classification of RDF, entity classification, and entity resolution. All these methods are done within the KG to improve its performance and diversity. Out-of-KG works externally using the KGE for relational extraction of information implemented with the distance of RECAL models, Recommender Systems to give user feedback based on similar users of an application, and finally Question-Answering by giving the KGE a question and getting the output as an answer. The latter could help the Explainability problem returning information about the decisions of an algorithm.

Bordes, Antoine, et al. [16] describe distance models, specifically translational embedding model 'TransE' for modelling multi-relational data. The results of the experiments of this method outperform RESCAL and Structured Embedding models on link prediction with minimal parameterisation. This approach is highly scalable and performs well on large datasets.

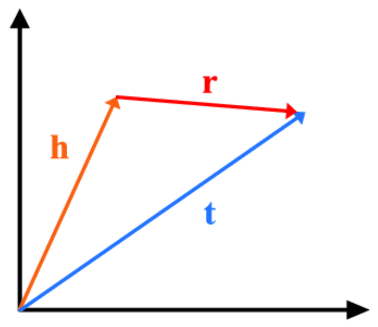


Figure 2 Simple diagram of TransE model [32]

Fan et al. [33] discuss a 'TransM' model which pre-trains the triplet's weights with a relational mapping property, as 'TransE' is "not flexible enough to tackle well with the various mapping properties of triplets". 'TransM' offers lower parameter complexity to 'TransE' and provides a more flexible and extensive mapping of the triplets in the KG.

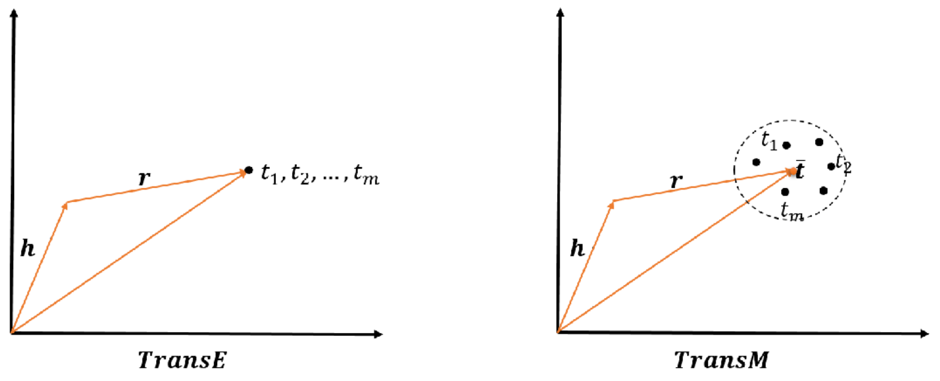


Figure 3 Comparison of TransE and TransM [33]

Wang et al. [34] describe the Translate on Hyperplane model 'TransH' which also assumes embeddings of entities and relations being in the same space. It attempts to improve on the 'TransE' model, which is terrible at modelling with 1-N, N-1, and N-N relations because of its scoring function. It provides improved performance with the capabilities of scaling that the 'TransE' model has. The 'TransH' paper [34] also uses a trick to reduce false-negative labels by setting "different probabilities for replacing the head or tail entity when corrupting the triplet" which is sufficient for real-valued incomplete KGs.

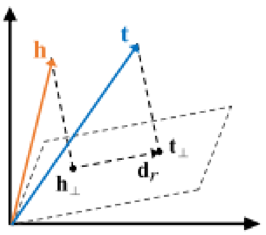


Figure 4 Illustration of TransH model [32]

Lin, Yankai, et al. [35] describe an improvement on both of these models with a 'TransR', that maps entities and relations in separate spaces to reduce the insufficient space problem, which 'TransE' and 'TransH' possess. These models can have entities nearby which have no relation, and therefore the 'TransR' offers a consistent, significant improvement over these models.

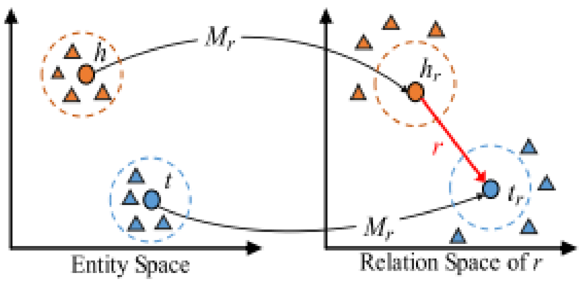
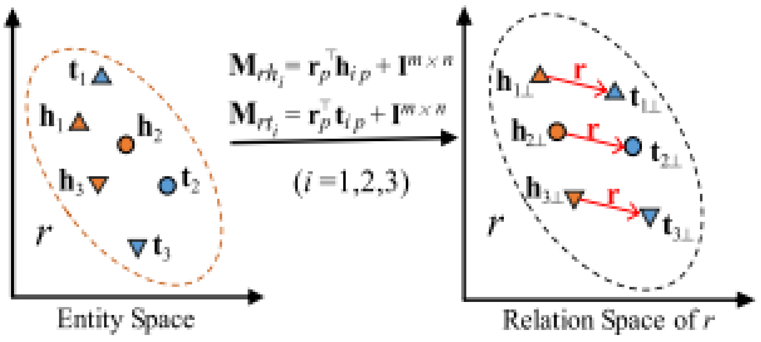


Figure 5 Mapping of TransR model [32]

Chang, Zhu et al. [32] discuss a 'TransD' dynamic translation model which aims to overcome the weaknesses of 'TransR'. When mapping entities, we find that the subject and object are usually different categories, thus should be transformed differently. 'TransR' has a considerable number of parameters, and due to this complexity will find it challenging to work with large-scale datasets. 'TransD' uses two vectors to represent each entity relation pair, the first describes the meaning of an entity/relation, and the second 'projection vector' create the mapping matrices.



t₁
h₁
h₂
h₃
t₂
t₃
r
$\mathbf{M}_{rh_i} = \mathbf{r}_p \mathbf{h}_{ip}^{\top} + \mathbf{I}^{m \times n}$
$\mathbf{M}_{rt_i} = \mathbf{r}_p \mathbf{t}_{ip}^{\top} + \mathbf{I}^{m \times n}$
$(i = 1,2,3)$
h₁⊥
t₁⊥
h₂⊥
t₂⊥
h₃⊥
t₃⊥
r
Entity Space
Relation Space of r

Figure 6 Mapping of TransD Model [32]

Yang et al. [36] propose the DistMULT model, which like RESCAL is a bilinear model but differs as non-diagonal entries in the relation matrices are assumed to be zero. Its results outperform RESCAL and TransE, with a similar number of parameters. Every entity and relation have a single latent feature vector which has a dimensionality depending on the number of nodes. It contains the same number of relational parameters as the 'TransE' model but with more exceptional performance on link prediction.

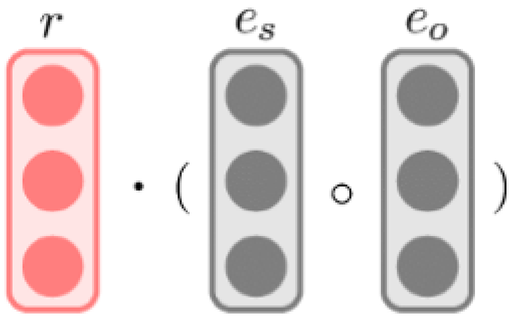


Figure 7 DistMULT vector calculation

2.4.2 Graph Feature Models

Graph Feature models extract features from the observed edges in KGs, and recent developments in rule mining models such as Association Rule Mining under Incomplete Evidence (AMIE) have shown their effectivity. Galárraga, Luis Antonio, et al. [18] describe AMIE a rule-mining model which can work their real-world datasets. The system requires only the knowledge base; no parameter tuning is needed and outperforms many leading KG models in terms of runtime, producing high-quality output rules. This model is further improved by Galárraga et al. [37], introducing AMIE+ to allow for scalability on massive knowledge bases, which speed up the refinement and confidence evaluation phases.

Similarity-based algorithms map relationships between entities neighbouring nodes or paths between nodes. The measure of similarity between entities has three categories local, global and quasi-local indices [17]. Local indices are independent neighbours entities which are fast to compute, making it an efficient, scalable approach, but it cannot learn patterns outside of the locality. Global indices operate with higher prediction accuracy but have more computation. Finally, Quasi-local indices attempt to bound global indices to reduce analysis while keeping accuracy.

Path Ranking Algorithms (PRA) are another type of Graph Feature model KGE. PRA extends from quasi-local similarity-based approaches to learn big multi-relational knowledge graphs.

Lao and Cohen [19] describe a method for learning the weights of path-constrained random walkers. This novel approach comes from a proximity measure Random Walk with Restart (RWR) which labels individual edges. The path-constrained process instead marks weights to particular sequence supported by 'query-independent experts' to generalise PageRank measure and 'popular entity experts'. These sequences adjust the PageRank for essential entities. PageRank makes it possible to rank the series in order, and adding this path constraint adds a crucial attribute of highlighting particular paths giving it the ability to learn sequences effectively.

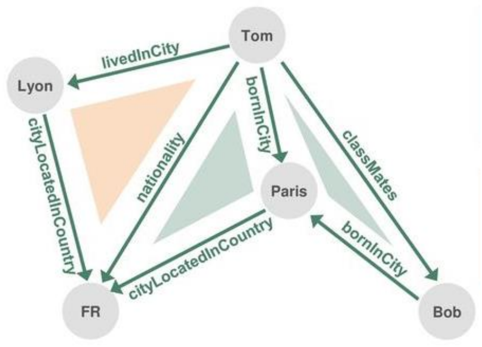


Figure 8 Path-Ranking Algorithm example [37]

Lao, Cohen et al. test the Path Ranking Algorithm further in 2011 [38] to show that soft inference in the combination of constrained, weighted, random walks can give new beliefs for a knowledge base, delivering high efficiency on large datasets. PRA is easily interpretable as it highlights the most critical paths as rules which are understandable to the user.

2.4.3 Other Techniques

Nickel, Jiang, and Tresp [39] attempt to combine latent and graph, feature models. Latent models are more effective with global relations that have new variables, whereas Graph feature models work better for modelling local and quasi-local patterns. This model is efficient and scalable for factorisation. It improves the predictive performance but also reduces the ranking leading to quicker results. It completes its full runtime with 3% higher accuracy than RESCAL before the second iteration.

Joulin et al. [40] construct a linear model to learn KGEs by casting graph problems into supervised learning as a non-relational approach. The disadvantage of this is that the dataset must be clean, and the task must require direct information from the graph. It does provide a rapid and relatively straightforward approach to learning KGEs.

Nickel et al. [41] propose a Holographic embedding (HolE) method for KGs. It is related to holographic models that use circular correlation to make compositional representations. HolE can discover high-quality interactions while maintaining efficiency. It provides improvements on SOTA models such as RESCAL, TransE and TransR and provides the ability to query results with Question-Answering.

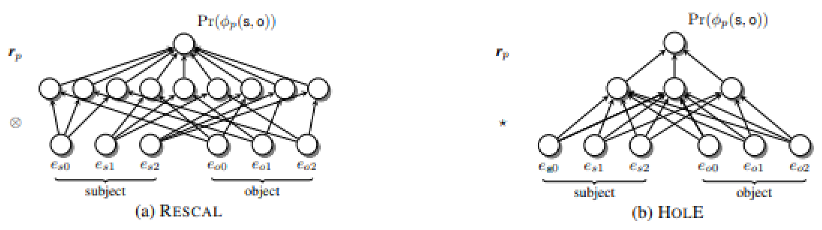


Figure 9 RESCAL vs HolE models [41]

Trouillon et al. [42] propose ComplEx embedding model which uses complex numbers to represent the embeddings, therefore, allowing symmetric relations. Similarly to HolE, ComplEx uses latent factorisation which "can handle a large variety of binary relations, among them symmetric and antisymmetric relations". Compared to other high-quality embedding methods such as HolE, it provides more straightforward computation using the Hermitian dot product between vectors while still producing very accurate results as a scalable model.

Trouillon and Nickel [43] devoted a paper to the comparison between ComplEx and HolE models. It shows that the two models are equivalent in terms of output but, produce discrepancies in results due to different loss functions. Both can train equally well on asymmetric and symmetric patterns. ComplEx has the advantage of time complexity which is linear compared to HolE quasilinear. HolE is advantageous over ComplEx though as embeddings stay in the real domain, which is easier to use for real-world applications.

Dettmers et al. [44] introduce a multi-layer convolutional network mode 'ConvE'. This approach has eight times the parameters of the shallow fast model DistMULT most KGEs but still achieves state-of-the-art results for its Mean Reciprocal Rank (MRR). Using simple models are more scalable for bigger datasets at the cost of learning less expressive features. Convolution operator is parameter efficient and fast to compute, due to highly optimised GPU, defined by three layers – single convolution, projection, and inner product layers. It is most similar to HolE as it uses circular correlation, which is the inverse of circular convolution, but HolE is less expressive. Compared to ComplEx and DistMULT, which produce MRR of 0.34

and 0.33, respectively, on a test YAGO dataset [24], the ConvE model scores a 0.44 with similar efficiency.

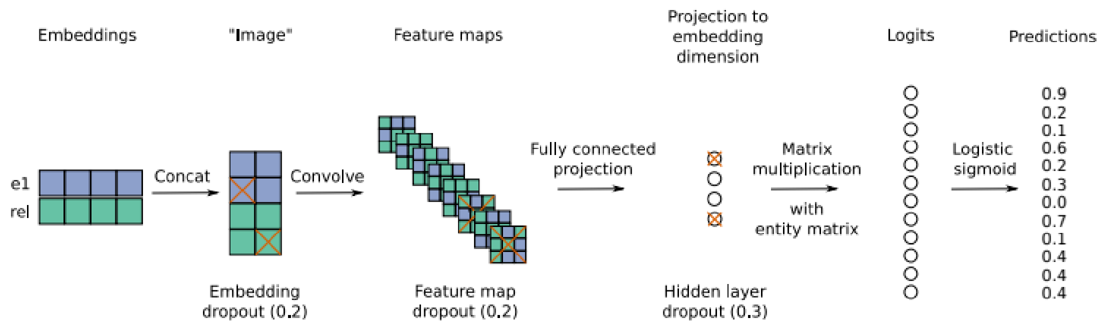

Embeddings
"Image"
Feature maps
Projection to embedding dimension
Logits
Predictions
e1
rel
Concat
Convolve
Fully connected projection
Matrix multiplication with entity matrix
Logistic sigmoid
0.9
0.2
0.1
0.6
0.2
0.3
0.0
0.7
0.1
0.4
0.4
0.4
Embedding dropout (0.2)
Feature map dropout (0.2)
Hidden layer dropout (0.3)

Figure 10 Demonstration of ConvE Model [44]

Nguyen et al. [45] propose a novel convolutional embedding model 'ConvKB' to capture relationships in a knowledge base. A three-column matrix represents each triple which is inputted into the convolution layer with multiple filters, concatenating to a single feature vector output. The feature vector is multiplied by weight to return a ranking score for link prediction tasks. 'ConvE' observes only local relationships among different dimensional entries, whereas 'ConvKB' takes global relationships among same dimensional entries of an embedding triple, which is very useful for KG completion tasks. During evaluation on two benchmark datasets, WN18RR [46] and FB15k-237 [47], 'ConvKB' scores an MRR of 0.396 outperforming 'ConvE' with 0.3116 on FB15k-237 [47] dataset but 'ConvE' scores 0.46 compared to 'ConvKB' 0.248 on the WN18RR [46] for MRR also, showing they both can be useful depending on the problem. It shows promise for future convolutional embeddings as even these pioneering models outperform many of the SOTA algorithms.

2.5 Knowledge Graph Completion

The three main approaches to KGC are Entity Resolution, Probabilistic Soft Logic, and knowledge graph embedding. Although section 5 uses the KGE method for testing link prediction, subsections, 2.5.1 and 2.5.2 will discuss the other approaches.

2.5.1 Entity Resolution

Entity Resolution (ER) is a method of link prediction, which attempts to group collective entities which link to an underlying entity. It is a less constrained method than data tables because KGs use multiple categories to represent entities. It is more effective than standardised representation because multiple KG can merge under different ontologies. To learn the groupings of entities, ER uses techniques of string matching of entities text, similarity measures, and blocking functions which are a configurable inverted index. The similarity technique determines similar entities, and the blocking process reduces extensive pairwise complexity, both of which are training using ML. The below diagram shows how using ER can reduce the complexity of a graph by grouping entities which are the same but labelled differently.

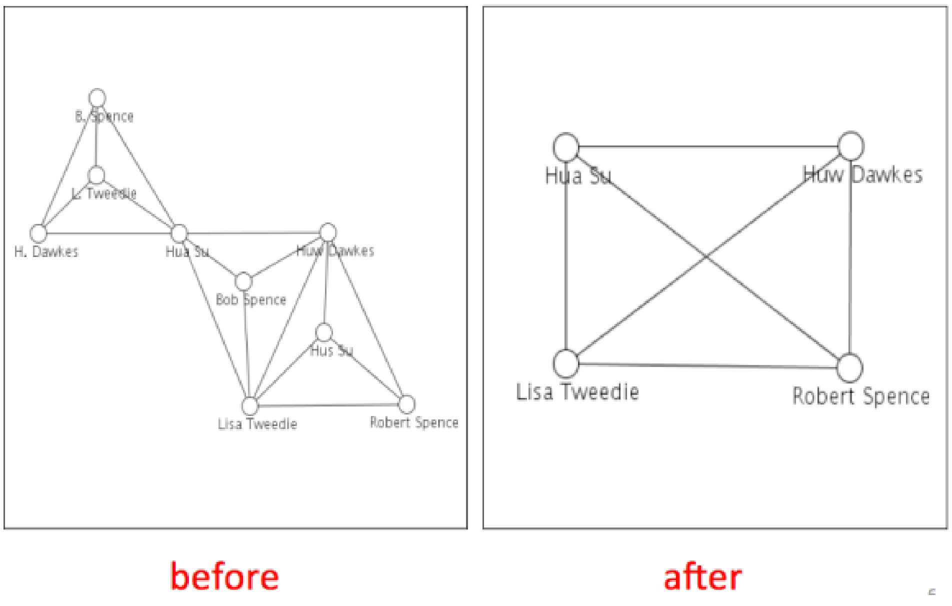


Figure 11 Entity Resolution Example [48]

Elmagarmid et al. [49] discuss different approaches to Entity resolution, also called duplicate detection. Similarity techniques, such as character, token, phonetic, and numerical matching. These are used in the various algorithms to compare entities and decide if they are the same. These algorithms vary from Bayesian decision rules, supervised and unsupervised ML algorithms, distance-based techniques, and rule-based techniques. They divide these methods into two categories: ad-hoc and principled methods. Ad-hoc algorithms work quickly and are more scalable, whereas principled methods are more accurate through probabilistic inference.

Kopcke and Rahm [50] discuss entity resolution approaches on real-world matching problems. They draw comparisons between state-of-the-art models for ER to understand which ones work best for specific tasks. To do this, they use an evaluation method called Framework for Evaluating Entity Resolution (FEVER) which tests the methods for precision, recall, and F-measure, and efficiency of runtime. Tests on non-learning ER models, such as COSY (Commercial System), PPJoin+, and FellegiSunter, and learning ER approaches, for example, FEBRL (Freely Extensible Biomedical Record Linkage) and MARLIN (Multiply Adaptive Record Linkage with Induction), on real-world datasets. The datasets used are benchmark academic KG and two e-commerce large scale datasets Amazon-Google Products and Abt-buy. None of the methods performs well on the e-commerce data because the algorithms were not sophisticated enough. For the academic data, COSY performs efficiently and effectively for one attribute matching beyond this, the quality of performance reduces. FEBRL and MARLIN both use support vector machines to learn the entity similarities, for matching more than one attribute these methods give the best performance in terms of F-score. PPJoin+ was the most scalable approach for single characteristics producing quicker results than COSY. Overall we see how effective ER can be even for real-world implementation, but there is still much to be improved upon before it produces results on par with KGE.

2.5.2 Probabilistic Soft Logic

Probabilistic Soft Logic (PSL) is a second method for link prediction built on an SRL framework for modelling probabilistic and relational domains by using probabilistic inference. It combines first-order logic and probabilistic graph models, to represent complex relations and determine incomplete KG, respectively. It creates accurate KG from noisy internet scraping and ER by using soft logic to scale truth values between 0 and 1. It is a scalable approach which can work with millions of entities and compute their truth values efficiently.

PSL uses joint reasoning to combine entities which are the same, similar to ER, but it converges the most likely extractions solved as a convex optimisation problem. PSL is different from other methods as is probabilistically infers relationships likelihood from current knowledge. Markov Logic Networks are the basis of PSL, but an MLN has the disadvantage only being able to take Boolean truth values for logical relations in triples. Therefore, there is no probability scale, just a true or false label. The soft logic which PSL uses gives it the advantage of confidence scores from newly predicted links which accept a defined threshold level.

A PSL operates by defining the input of specific rules and positive triples from a dataset. For each pairing, a soft truth value is calculated based on the likelihood of two linked entities. Each truth value must verify against the rules. By confirming the head and tail of a new triple are within a threshold defined by the distance between entities, demonstrates its likelihood. PSL has prediction applications of missing links or further information based on current knowledge using ‘Maximum A Posteriori’ (MAP) inference. MAP calculates the most likely unobserved entities based on soft truth values. The outputted probabilities decide whether a fact gets added to the KG as a newly predicted link. Brocheler et al. [51] introduce PSL as a method of reasoning through similarity for relational entities. PSL ranks higher for F1 score in comparison to other state-of-the-art ontology alignment systems, with a score of 0.865, showing it can accurately learn entity pairs.

2.5.3 Knowledge Graph Embeddings

Depending on the source of the data, KGEs can be the most effective method for KGC, but KGEs do not perform well on sparse, noisy KGs that are extracted with text-mining. Finding missing entities is similar to the approach for evaluating the embedded models, by way of ranking real triples in order of probability. The goal of prediction is to improve the quality and variety of triples in the KG, to essentially adding all possible connections between entities. KGE approaches vary depending on the model chosen. Examples include TransE (distance-based), ComplEx, and HolE (both semantic matching-based).

Lin et al. [35] compare translational models (TransE, TransH, and TransR), on link prediction tasks for large benchmark datasets, Freebase [47] and WordNet [46]. Each KGE model ranks sets of possible entities for missing links in the graph and compare their performance by Mean Rank and Hits score to determine if these rankings are accurate. TransR outperforms both TransE and TransH in terms of Mean Rank and Hits@10 score for link prediction for the benchmark datasets, along with several other models such as RESCAL. On the WordNet [46] example, all SOTA models achieve a Hits@10 score of over 80% with all TransR models achieving over 90%, showing KGE effectiveness for this task.

Trouillon et al. [52] describe the best method to use the ComplEx model for KGC. They tell how using complex tensor factorisation can be advantageous as it provides greater detail given to the embeddings. It has the requirements to implement good KGC, as it is scalable and capable of dealing with a large variety of relations. They compare the MRR and Hits@ rate for this model in comparison to other SOTA models to show a vast improvement proving that complex factorisation is effective and efficient for KGC.

Hayashi and Shimbo [53] further reinstate Troullion and Nickel’s [43] work on the comparison between their models ComplEx and HolE. Hayashi’s paper focuses on the comparison of these models for link prediction tasks. Through their [53] evaluation, they reveal that the ComplEx model can be viewed as an instance of a holographic model as they produce the same scores for ranking triples. It determines that all complex entity embeddings can transform into holographic embeddings. Primarily, the ComplEx and HolE model operate very similarly and produce almost identical results.

Xiao et al. [54] build a model, ManifoldE, for precise link prediction. They determine two reasons why accurate link prediction is a difficult task; poorly defined systems and restrictive geometric format. The ManifoldE model creates the embedded vectors either spherically or through a hyperplane, depending on the problem. The models achieve a significant improvement on other models for Hits rate, tested with Freebase [47]and WordNet [46] datasets. The spherical method achieves the most accurate rankings (Hits@10: 94% WordNet and 79% Freebase [54]) and is more efficient in terms of runtime.

2.5.4 Open-World KGC

The previous three examples all use a closed-world assumption, i.e., KGs are fixed and adding new entities is difficult. Recently, there is growing research for an open-world approach, as discussed by Shi and Weninger [55]. They introduce a new model called ConMask which learns embeddings of the entity's name and features to connect unknown entities to the KG. Closed-world KGC works best when predicting relations between known entities which have many connections already. For entities with few links, these methods perform poorly, which is what ConMask aims to improve. Convolutional Neural Network (CNN) trains the ConMarks model, integrating related text and using relationship-dependent content masking to reduce noise from an entities description. Finally, it finds a similarity score between entities from their embeddings utilising a form of targeted ER. The below illustration, Figure 12, shows how ConMask operates taking the entities name and description from head and tail, with the relation, and using this as input data to the CNN to output predictions.

ConvMask outperforms other open-world approaches, such as 'Target Filtering Baseline' and 'Semantic Averaging', for benchmark datasets in terms of MRR and Hits@rate. Using the DBPedia500k [56] dataset, ConvMask scores 0.33 for the head entity and 0.47 for the tail entity for MRR metric, compared to 'Target Filtering Baseline' 0.01 for both and 'Semantic Averaging' 0.09 and 0.13 for head and tail respectively. Hits@10 rate is far above the others also with 0.17 for head and 0.52 for tail compared to 'Target Filtering Baseline' 0.01 head and 0.02 for the tail, and 'Semantic Averaging' 0.04 and 0.16 for head and tail respectively.

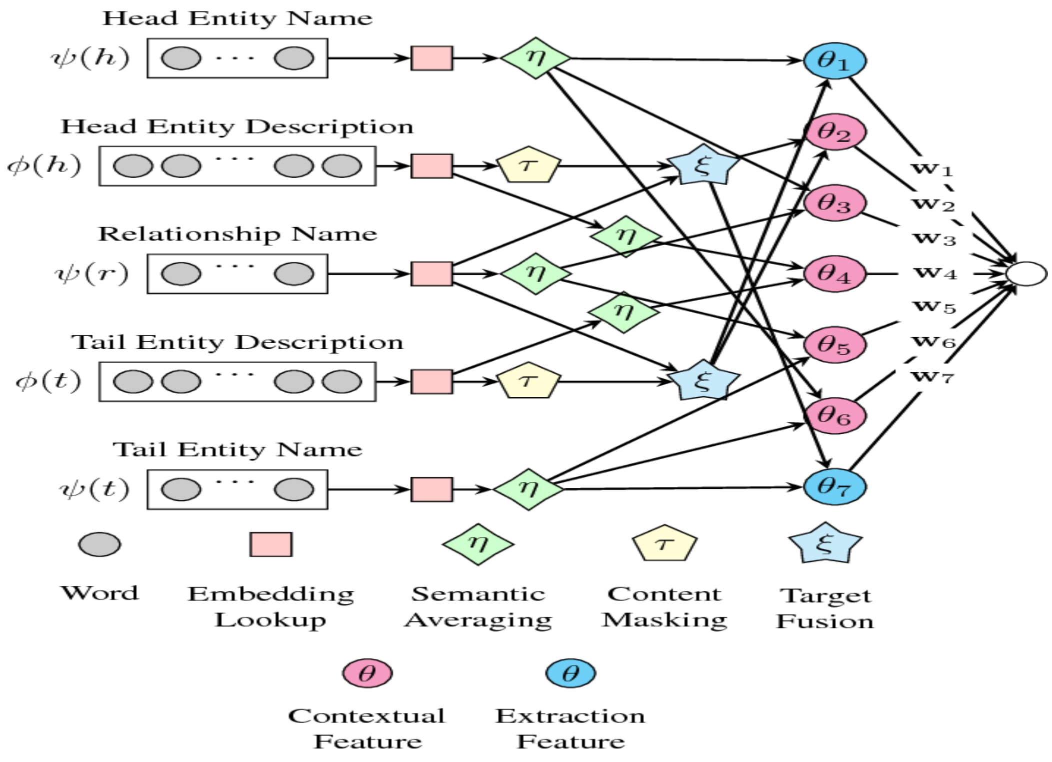


Figure 12 ConMask model for Open-World [55]

This model even outperforms closed-world approaches for closed-world tasks. It is compared [55] with the Translational KGE models TransE and TransR. When tested on DBPedia50k [56] dataset, TransE and TransR produced Hits@10 score of 0.68 and 0.67 respectively compared to ConMask's score of 0.72 for the tail entity, and 0.37 and 0.39 score respectively compared to ConMask's 0.41 for the head entity. The DBPedia50k [56] dataset generates a sparse graph and is more difficult for other closed-world KGC methods. Although the ConMask did not behave well on other closed-world task datasets, such as FB15k [47] and DBPedia500k [56], compared to these models, it is adequate as these tasks are not ConMask's intended objective.

2.6 Real-World Applications

Wang et al. [5] use a KG with a Bidirectional Recurrent Neural Network (BRNN) to retrieve more information about a biomedical literature-based discovery. BRNN is a type of Neural Network which connects hidden layers in opposite directions for the same output used to increase the amount of information in the network by allowing future input data available at the current node. Doing this involved four steps – construct the KG by extracting relations between entities from the biomedical literature source, then converted into a low-dimensional vector space, created a bidirectional Long Short Term Memory (LSTM) was trained based on the entities relations. Finally, the model can complete discovery tasks. The results prove that the method capably discovers links between entities and their interactions, meaning it can capture complex associations between unknown entities. The conclusion of this report illustrates the advantages of combining KGs and Deep Learning models.

Catherine et al. [57] discuss the possibility to use KGs to explain to user's decisions choices using a page ranking method ProPPR. ProPPR uses a set of rules to deploy based on the relations in the trained KG. The model then evaluates for movie rankings to give user feedback, in the form of detailed explanations.

Wang et al. [58] describe a more sophisticated KG based recommender system Path Recurrent Network (KPRN). KPRN exploits extra user-item connectivity by comparing semantics of the entities and the relations in the KG. It substantially outperforms similar state-of-the-art models because the path extraction produces better ranking scores, giving informative explanations for decisions. This model enhances the Recommender systems greatly as it has the ability of reasoning and Explainability.

Goebel, Randy, et al. [59] examine current applications of AI and how they can be improved by the implementation of knowledge bases to include Explainability in the model. The paper depicts a valuable dataflow diagram of how this project can implement Explainability with machine learning algorithms by adding a KG.

The problem of Explainability in KGEs is now better understood through this detailed study of methods and specific techniques for achieving high quality, scalable, and fast runtime results. The next phase, of the project, will describe the creation of a knowledge graph network from a dataset. From there, it will be possible to produce an embedded vector output, testing some of the above algorithms. The final phase of this research will choose the most applicable model for the dataset, which will further investigate knowledge graph competition and interpretability for the user.

3 Knowledge Graph

3.1 Overview

Chapter 3 describes the raw dataset and the pre-processing steps which transform the data into a KG, capable of being trained as an embedded model. Section 3.2 outlines the type of dataset chosen, its contemporary significance to research, and why it is suitable to be represented as a knowledge graph. Section 3.3 reports the transformation of the data into a KG, discussing the source, structure, and code written. Section 3.4 depicts an enhancement of the KG completed through the use of filtering to increase its ranking performance as an embedded model. The final section of this chapter 3.5, illustrates the creation of a visual portrayal of the KG model and discusses the benefits of the Python library used to create it.

3.2 Dataset

A knowledge graph maps relationships from data into an ontology, therefore a sizeable semantic dataset is needed for experimentation. The larger the dataset, the more accurate predictions of the knowledge graph embedding. The KG will represent the interlinking of entities found in a database, examples of entity classifications include events, locations, situations, objects, and concepts. These entities must have a strong correlation to each other, so they can form a network, and be described in a formal structure for efficient processing and comprehension to the user. The relationship between these entities can be tagged as relationship types, for example, institution, Author, citations. Not all RDF triple datasets are knowledge graphs, to make this assumption the RDF datasets must represent semantic knowledge of the data. Applications of KG vary for different fields of study. However, they are most effective for information-heavy services such as medical research papers, semantic searches, content recommendations, and investment market intelligence. The first challenge of this project is finding a dataset which contains dense, semantic content with linking relationships between each node.

In response to the novel COVID-19 virus, there have been thousands of research papers released to gain a better understanding of the disease. An open dataset, COVID-19 Open Research Dataset (CORD-19), has been created by the Allen Institute for AI partnering with a coalition of leading research groups contains over 192,000 scholarly articles, 84,000 of which are full-text, about COVID-19, and related coronaviruses [60]. The purpose of this growing dataset is to allow a global research community, merging the biomedical and AI research communities, to help develop text-mining and information retrieval from the metadata to combat this pandemic. The papers have been sourced from highly-respected peer-reviewed research groups, PubMed, the World Health Organization's Covid-19 Database, and preprint servers bioRxiv, medRxiv, and arXiv. There are various research directions which one could make use of the dataset outlined in [60] predominately for clinicians, clinical researchers, and NLP (Natural Language Processing) research. The use of information retrieval, text classification, and knowledge graphs are listed as possible avenues to analyse the dataset. There have also been many online challenges listed on Kaggle [61], a data science community, to encourage more research for different open-ended questions, with the use of CORD-19, with the incentive of prize money for the winning participants.

Although this dataset sufficient for this project, there are some limitations to discuss. Only specific document types of research papers and preprints are used in CORD-19, leading to potential incomplete information. Foreign languages, such as Chinese, are under-resourced in the dataset which, especially during the early stage epidemic, could provide valuable substance.

The overall conclusion of the paper [60], challenges researchers to use AI to further advance scientific developments by providing easily accessible and linked information. The paper also notes the importance of shared resources, mentioning that the collaboration between different sectors can lead to more efficient solutions. It is vital to gain a better understanding of the virus, through academic research, as the COVID-19 pandemic is of critical consequence to society. By using CORD-19 for this project to test knowledge graph embedding models, it may help the research community.

3.3 Creation of KG

CORD-19 was openly available, updated weekly and stored in Comma Separated Values (CSV) file format. There are 12 attributes for each paper detailed in every row of the file. The important ones include a unique ID, source, title, year, and journal name. Transforming this file to a knowledge graph needs the conversion of information into RDF triple format, extracting the entities and relations from the dataset. Amazon Web Service's (AWS) pre-built Covid-19 knowledge graph (CKG) [62] replaces the need to slowly and inefficiently extract the related entities through data wrangling, which would affect the project's timeline. CKG uses the CORD-19 to make connections between scholarly articles, authors, scientific concepts, and institutions using predefined edges. To create entities, the CKG is built using Amazon Neptune, a fast and reliable graph database service, and incorporates annotations from Amazon Comprehend Medical. This NLP service can extract relevant medical information from unstructured text. CKG is a directed graph meaning that before its creation, it has constructed the relations between source and destination nodes. The CKG contains six different edges to describe the relationship between nodes.

1. affiliated_with – Relationship between Author and their Institution
2. authored_by – Relationship between paper and its Author
3. associated_concept – Relationship between paper and related concepts, weighted with Amazon Medical confidence scores
4. similarity – Similarity between two papers based on the crossover in text
5. cites – Relationship between papers cited in another paper
6. associated_topic – Relationship between Papers and 1 of 10 labelled topics

The data is in CSV format, but there is a separate file for each of the relation types listed above and for each of the five node types - paper, Author, institution, concept and topic. These contain further information about each node. Two JSON files also store data about each papers title, and identification number respectively, which can provide additional explanations. Each edge file contains the identification numbers of the source and destination node and the relation between them. The files are merged into one to create the full KG, with every link in RDF triple format – (source, predicate, destination). As these files are large, it is most efficient to merge them with a short script using one of Python's (programming language) packages 'Pandas'.

The six edge CSV files are stored Google Colab, a cloud service for coding Python through a browser. The identifier column used to label each relation was removed, as it was unnecessary for the knowledge graph. The column titles were renamed and reordered to "subject", "relation", "object", and "score". The subject and object columns contain the unique node identifier, and the relation column contains the edge between the two nodes to form the RDF triple. The final column "score", was derived for 3 of the relations, namely the concept, topic, and similarity edges. The Concept score is determined using ML from the Amazon Comprehend Medical, indicating the probability that a concept is in a paper. Latent Dirichlet allocation (LDA) is the method used to calculate the Topic score, which is a statistical model that groups nodes based on observations to categorise similar nodes, the score denotes the probability a paper relates to a topic. Finally, the Similarity score determines if two documents are related to each other. The method to calculate the Similarity score is the weighted sum of all the paths between two papers generated for all documents and ranked. These three scores can provide weightings to edges in the knowledge graph embedding for Explainability. The six files were concatenated and saved as a separate "dataset.csv" file, stored in google drive for ease of access, with 2,771,525 rows to describe all the relations in the dataset.

Five node CSV files are detailing the attributes of each entity, namely concept, institute, Author, paper, and topic. Each entity has a unique identifier which relates to the actual name of the entity. This relation acts as the outer layer of the knowledge graph, which to help to explain more about each entity. Similar to the data wrangling of the edge CSV files, Pandas is used to extract the RDF triples associating each identifier with its name with its relation as the link. The institute file was missing the name of many universities, and therefore the county of origin was concatenated if none was present. The script removes all triples, which contained empty entities because they were futile for the information gained from the network. The code merges these files and saves it as a new CSV file "dataset_entities.csv". Finally, another script combines it with the "dataset.csv" to a new CSV file, totalling 3,168,262 triples, saved as "full_dataset.csv". This knowledge graph is convenient for extracting information quickly and efficiently from the multi-sourced dataset. This heterogeneous KG must be converted to an embedded model so information can effectively be learned about the data to generalise its context and to infer relations. Different KGE models will be tested in chapter 4 to determine the best approaches but also to discover more about the dataset.

3.4 Filtering

After the completion of some knowledge graph embedded tests which resulted in poor performance, a filtering phase to enhance the KG was necessary. There were over 780,000 unique entities, many of which only appeared once in the KG. To effectively train the embedded model, a web of related entities are needed to learn the features of the graph. Having many entities within the triples, which are not frequently reoccurring, has a significant effect on the performance of the model. A new rule for the dataset was needed to solve this issue, which involved the removal of triples that contain entities (both source and destination) which do not occur at least five times. Accomplishing this task involved the creation of a python script, which calculated the total frequency of each subject and object entity, across all entities, for every row in the dataset. If this frequency was less than five for either subject or object the full triple was removed from the dataset. The script cleared the indexes and shuffled the dataset to reduce biases. As a result of this implementation, the dataset size was reduced dramatically to 2,343,610 with only 138,309 unique entities. This filtering procedure reduces the processing power required to create KGEs, as there are less embedded vectors needed. Although a smaller dataset means there will be fewer links in the knowledge graph, the embedded model will be much more effective at prediction because it does not learn from entities with insufficient relations. The dataset is now ready to be tested again on the knowledge graph embedded models, with the hope of improving the evaluation results.

3.5 Visualising

Interpreting the knowledge graph's structure further required a script to help visualise the connections between the entities. A Python library called NetworkX created the KGs structure. NetworkX is used to map complex networks, allowing the ability to make, manipulate, and examine its shape and size. It contains several functions to allow for customisable graphing and analysis. The program created a graph object, which is converted from a Pandas dataframe with a simple built-in process. Here the columns for subject, object and relation are defined along with the type of KG to be built. There are different categories of networks, depending on the dataset. Directed graphs represent when the direction of the relation attribute matters between pairs of entities. The second subcategory is if the system is allowed to contain multi-edges between each pair of nodes. This project's Covid-19 CSV dataset required both of these, defining the network as a Multi Directed graph. NetworkX can also weight a plot based on the significance of edges, but this is used predominantly for numerical datasets. It is possible to weight edges and nodes for graph generation and learn about its statistics, but what is useful for this project is a module called 'Draw'.

NetworkX's primary purpose is not to plot graphs. However, it contains the package for drawing which can be used with a visualisation library, such as Matplotlib, to create an accurate depiction of the network itself. The graph object, called the 'Draw' function allows customisation for creating a plot of edges and nodes in terms of colour, transparency, and information. A simple graph, made from a sample of 200 triples, which transformed the KG into a graph object, shown in Figure 13. The colour of each edge is set based on the 11 different relation types in the knowledge graph. All entities are the same sky blue colour, with their names shortened for visual effect.

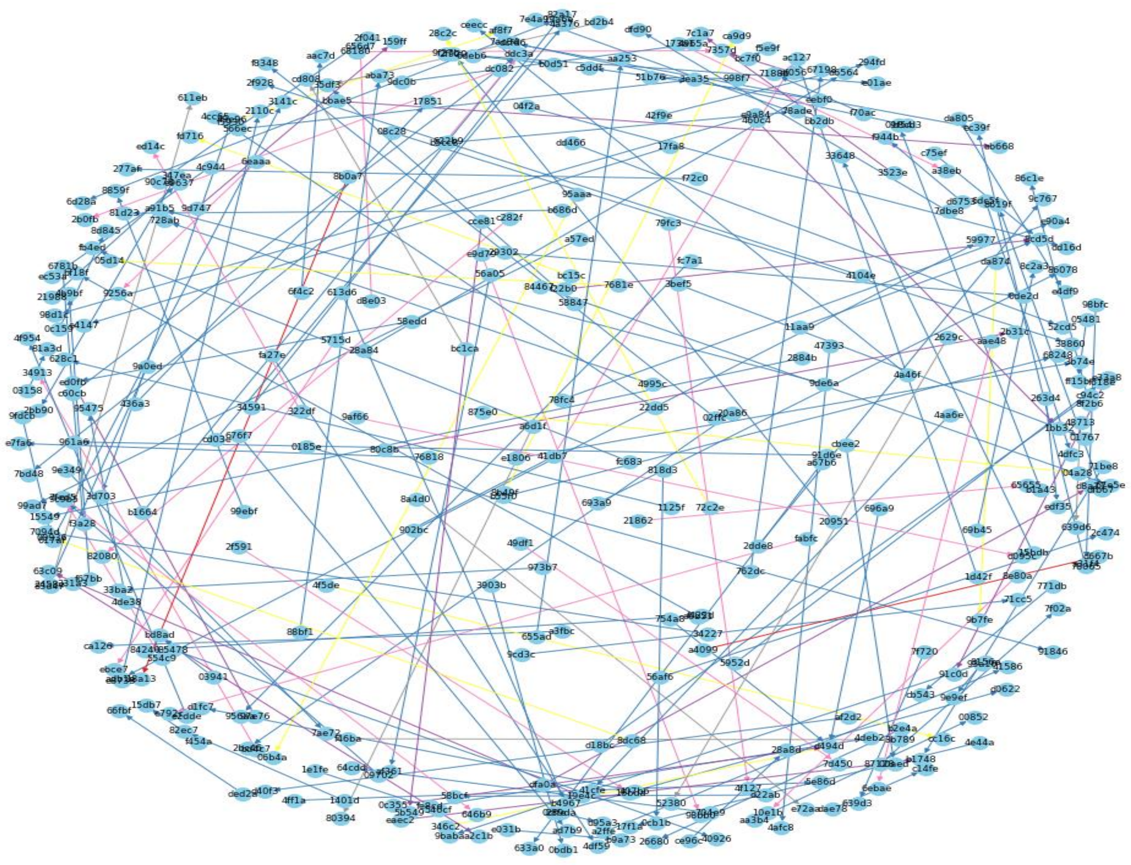


Figure 13 Sample of 200 from the Covid-19 Knowledge Graph

Although this plot looks impressive, it is not very informative. Any part of the KG can be subdivided from the dataset and displayed as a zoomed-in plot. The entity "SARS" and some of its connected entities for the topic, concept, and name relations were extracted from the dataset and converted into a second plot (Figure 14). This graph sets the colour of the edges based on the type of relation, but there is an increase in node size to show a more focused view of the connections. The relation labels are added with small font size and transparency so as not to clash with node labels. The node labels are made more prominent, and the full title of each node is displayed. Three papers labelled 'A', 'B', and 'C' on the plot link to the "SARS" entity, each also connect to two topics. From the plot, the conclusion that papers 'B' and 'C' are the most similar as both link to the same topics – "public-health-policies" and "epidemiology", whereas paper 'A' connects to "pulmonary-infections" and "clinical-treatment". A KGE model could rank these two papers closer together than paper 'A' if used for a searching task. This type of modelling with NetworkX will also be beneficial for visualising

KGC in section 5, for example, displaying the KG before and after links prediction to demonstrate a complete graph. As seen below, the majority of entity titles are long unique identifiers which have no semantic value for users. However, the visual plot can provide some explainability for the predicted links the embedded model makes.

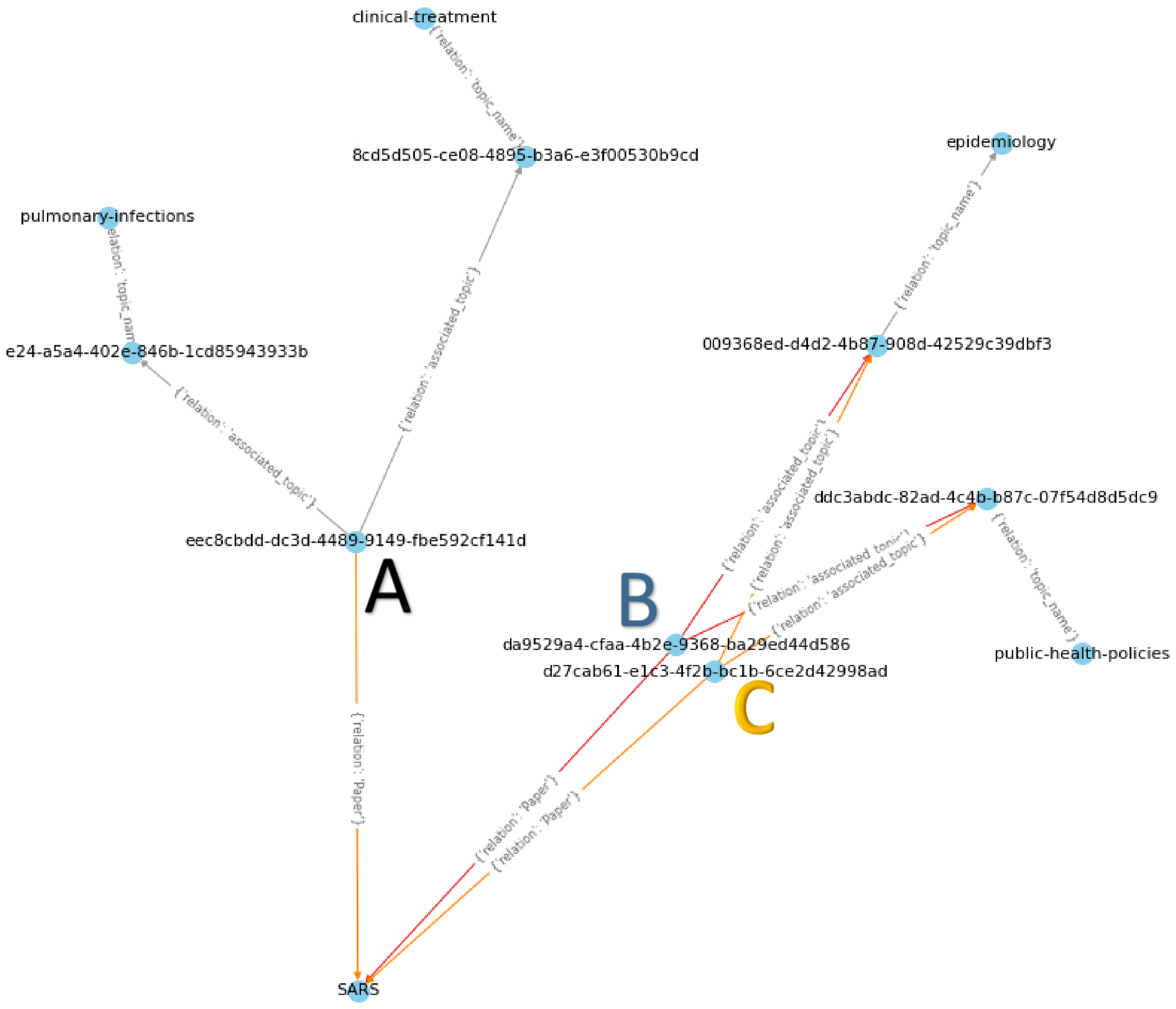

clinical-treatment
8cd5d505-ce08-4895-b3a6-e3f00530b9cd
pulmonary-infections
e24-a5a4-402e-846b-1cd85943933b
eec8cbdd-dc3d-4489-9149-fbe592cf141d
epidemiology
009368ed-d4d2-4b87-908d-42529c39dbf3
ddc3abdc-82ad-4c4b-b87c-07f54d8d5dc9
da9529a4-cfaa-4b2e-9368-ba29ed44d586
d27cab61-e1c3-4f2b-bc1b-6ce2d42998ad
public-health-policies
SARS
A
B
C
{'relation': 'topic_name'}
{'relation': 'associated_topic'}
{'relation': 'Paper'}

Figure 14 Zoomed plot of entities and relations from the KG

4 Embedded Model

4.1 Overview

Chapter 4 details the creation, testing, and evaluation of knowledge graph embedding models. Section 4.2 is a short section to discuss the SOTA models which the literature review (section 2.4) introduced. To create the KGE models, section 4.3 explains how an open-sourced library 'Ampligraph' assisted their training and the usefulness of this library's functions. The last section of this chapter, section 4.4, describes the examining and assessment of various KGE models to discover which one produces the best predictions.

4.2 Types of knowledge graph embeddings

A knowledge graph embedded model embeds the entities and relations of a KG into continuous vector spaces, to simplify its manipulation while keeping the original construction of the KG. KG embedded models set different score methods to calculate the distance of two entities and their relation in a format which is low-dimensional. This scoring method can then train the model, so that closely related entities are connected near each other, and entities not connected are far away. KGEs have many benefits, the most effective being its applications such as knowledge graph completion and Explainability from the extracting of relations. There are many different ways to create KGEs, which are described in the literature review (section 2.4). Fundamentally, by creating a vector for each entity and each relation, to capture the semantic properties of the knowledge graph. Distinguishing between KGE creation methods can be broken into three categories:

(i) The choice of representation of entities and relations – real numbers, matrices, or complex vectors
(ii) The scoring function – how the likelihood of the triple is estimated
(iii) The Loss function – minimising the objectives effect on training

Variations of these features lead to different models, which divide into the subsections, latent and graph feature models, or a combination of these. Latent feature models such as RESCAL and TransE capture the structure which creates the observed properties of a set of objects. Graph Feature models such as associated rule mining and PRA extract features from observable edges. Combined approaches, like ComplEx and HolE, can produce the best results in terms of accuracy. Novel convolution approaches such as ConvE and ConvKB can find more expressive features for embedding vectors while maintaining fast runtimes. Several models will be created and compared by their evaluation metrics in section 4.4. From these tests, the best performing model will then perform different applications in future chapters.

4.3 Creating a Model

For comparison, four models will be tested on performance to determine which is the most effective for CORD-19. This testing will also involve hyperparameter tuning to find the optimal combination of these for each model. Once the knowledge graph is in the correct RDF triple format, like the "full_dataset.csv" file, there are many ways to create an embedding. Developing an algorithm from scratch is a very time-consuming and challenging task, especially when aiming for high quality in prediction, therefore using a model pre-built model is the best option for this project. There are many libraries available which can load models with the ability to tune their parameters. When deciding on which one to choose it is essential to consider the runtime speeds, ranking scores, and most importantly, the variety of tools available for testing. Examples of standard tools include Python's Pykg2vec, Graphvite, and Accenture's Ampligraph. These libraries all produced competitive runtimes and accuracy results for their models. Ampligraph had the most algorithms available with the best tools for ease of creation and evaluating, which section 4.3.1 discusses below.

4.3.1 Ampligraph

Ampligraph is an open-source Python library that predicts links between concepts in a knowledge graph [63]. It has many functions for the creation and evaluation of knowledge graph embeddings. Ampligraph converts knowledge graphs in CSV or RDF triples format into vector representations. It then combines these vectors with scoring functions to accomplish various tasks such as link prediction and evaluation.

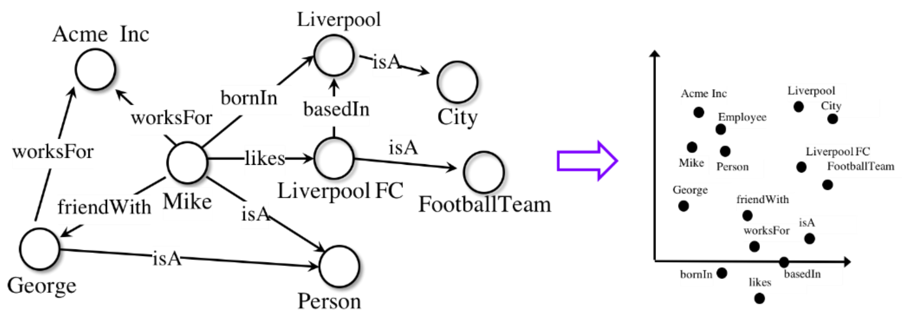


Figure 15 Ampligraph's Mapping of Entities to Vector Space [63]

It is built based on Tensorflow, an open-sourced platform for ML, which has a multitude of tools and libraries for the development conveniently make and deployable ML applications [64]. The design of the system is to run efficiently on CPU, GPU, and TPU processors. It includes five submodules which contain functions to make and test the KGE:

(i) Datasets – To load custom and benchmark datasets
(ii) Models – Prebuilt standard model such as TransE, DistMULT, ComplEx, HolE. Which allow the customisation of their hyperparameters
(iii) Evaluation – Functions for splitting the dataset, evaluating models, and scoring metrics
(iv) Discovery – Functions to find more about KG (finding clusters, duplicates and querying)
(v) Utils – Methods to save/load models and visualise embeddings

The benefits of Ampligraph for this project are its vast array of functions which are accessible and powerful at creating high-quality embeddings for large-scale knowledge graphs. Ampligraph's website has published runtime and predictive accuracy results from standardised tests. The runtime on 16GB GPU boasts under 1.5 seconds per epoch for all models with a large benchmark dataset (FreeBase 15K-237 [47]) for KG. Its metrics for the same dataset report a mean rank of 184-1024, mean reciprocal rank 0.23-0.33, and hits score (dependant on the number of hits) from 0.22-0.5 for all algorithms [65]. The high results recorded from these embedded models are equal if not better than comparable libraries, such as Pykg2vec and Graphvite. Ampligraph's short runtime, accurate results, and useful libraries are the reasons this tool will work well for all experimentation and testing of KGEs.

4.3.2 Hyperparameters

A hyperparameter is a parameter of an algorithm used to control the learning process. Hyperparameters differ from normal parameters, e.g. node weights, which are defined by the algorithm during training. Hyperparameters fall into two subsections, model hyperparameters refer to the model selection task, and algorithm hyperparameters have no influence on performance but determine the speed and quality of learning, e.g. batch size. Depending on the model, there are different available hyperparameters. KGEs learn by training a neural structure of a graph, which try to minimise the loss function the get the highest scoring function results.

There are five primary hyperparameters which can be set for KGE models in Ampligraph denoted the subcategories loss function, score function, regularizers, initializers, and optimizers.

(i) Loss functions:

Evaluates the solution to the objective function, which in ML algorithms minimises the error. The type of loss functions includes binary cross-entropy, negative log-likelihood (NLL), and max-marginal loss. The choice of loss function depends on the algorithm and its activation function.

Binary cross-entropy (BCE) measures the error between two probability distributions, for binary classification problems. The equation is given as:

$$L(y) = -\frac{1}{N}\sum_{i=1}^{N} yi \cdot \log\big(p(yi)\big) + (1 - yi) \cdot \log(1 - p(yi))$$

NLL uses the softmax activation function, used to compute the normalized, between 0 and 1, an exponential function of all the units in the output layer. It is most useful for classification tasks with binary or multi categories, given as:

$$L(y) = -\log(y)$$

Max marginal loss is using a margin to compare samples representations distances. A cost function selected to train only on the points where a data point is difficult to classify. It works most effectively with binary class problems, given as:

$$L = \begin{cases} d(ra, rp)\ if\ positive\ pair \\ \max(0, m - d(ra, rn))\ if\ negative\ pair \end{cases}$$

(ii) Score functions:

Ampligraph's models have defined scoring functions that combine the subject and relation of the triple with the object. Each algorithm has a different means of calculating this function:

TransE – BUGD+13 [16] relies on the distance between subject and object vector embeddings to calculate their similarity. Accomplished using L1 or L2 normalisation.

$$f = -\,\|es + rp - eo\|n$$

DistMULT – YYH+14 [36] is a trilinear dot product equation multiplying the distance between entities.

$$f = \langle rp, es, eo \rangle$$

ComplEx – TWR+16 [42] extends the DistMULT but with the Hermitian dot product by representing embeddings as complex numbers.

$$f = \Re e(\langle rp, es, \overline{eo} \rangle)$$

HolE – NRP+16 [41] use circular correlation to make compositional representations.

$$f = wr \cdot (es \bigotimes eo)$$

ConvE – DMSR18 [44] uses convolutional layers to calculate the embedding vectors.

$$f = \langle \sigma(vec(g([es; rp] * \Omega))W)eo \rangle$$

ConvKB – NNNP18 [45] similar to ConvE, this model uses convolutional layers but uses a dot product to find the vectors.

$$f = concat(g([es, rp, eo]) * \Omega) \cdot W$$

(iii) Regularizers:

Prevents overfitting occurring when training an algorithm with sparse predictors. It will also establish estimates if there is collinearity in the data. A regularizer objective is added to the loss function to minimise both while training which accomplishes collinearity. There is a higher penalty given to larger weights more if the regularizer attribute is substantial. Two forms of this are L1 and L2 regularization. L1 gives more sparse estimates which are an advantage over L2, as it enables the feature selection possibility and deals with high dimensional data more efficiently. The function provided by Ampligraph supports both L1 and L2 regularization.

For convolution methods, ConvE and ConvKB, Dropout regularization is also an option in Ampligraph. Dropout Layers sets random activations to zero to avoid overfitting. Dropout is applied to the outgoing edges of hidden units at each update of the training phase, selecting the neurons to use at random. Overfitting occurs when the model fits the training data too well reducing the loss of the training results but increasing the loss on the validation and test data. The Dropout layer has the effect of making the training process noisy, forcing nodes within a layer to take on more or less responsibility for its inputs probabilistically.

There is a third option to set the regularizers 'p' value to 3. This setting can be used for the ComplEx model to allow an improvement with the aid of canonical nuclear 3-norm. Lacroix at al. [66] propose the ComplEx-N3 model to enhance the standard ComplEx. Tensor nuclear p-norms is the basis of this regularizer, reformulated to solve the problem that makes predicates, or their inverse, fixed to their decisions in a dataset. Canonical Decomposition produces results far below state-of-the-art models but combining CD with ComplEx has shown improvements. For example, from benchmark dataset (FB15K [47]) tests the MRR of Canonical Decomposition and ComplEx are 0.40 and 0.80 respectively, but when combined, the MRR is 0.86. In section 4.4.1, there will be a comparison between ComplEx and ComplEx-N3 results, by changing the regularizer parameter to 'p=3'.

(iv) Initializers:

These initialise the embedding values before training which accelerates training but also prevents the layer activation outputs from exploding or vanishing during training. Vanishing gradient is where the cost of a gradient for cells becomes so low with an asymptotic function that it becomes very challenging to learn. In contrast, Exploding gradient gets stuck at a large number and cannot learn effectively either. Ampligraph provides three options for initialisation because if the neuron's weights are all zero, they learn the same features, which leads to undertrained models. The optimal method depends on the dataset and type of model chosen.

Random Normal – Initialises from a normal distribution with a specified mean and standard deviation. The mean is the average of all weights, initialised to zero. The standard deviation expresses how much the members of a group differ from the mean value for the group. The variance is the square of the standard deviation, set based on the number of nodes present. The bias, also set to zero, is the average difference between the predicted and actual value. High Bias leads to undertraining.

Random Uniform – Initialises the vectors to a uniform distribution with a specified high and low as bounds for range for values.

Xavier – Xavier initialisation makes the variance of the outputs of a layer to be equal to the variance of its inputs. It initialises to a uniform distribution or normal distribution depending on its parameters.

(v) <u>Optimizers:</u>

The purpose of the optimizers is to learn the optimal embeddings, so the model scores high on positive statements and low on unlikely statements. The most common optimizer type is an algorithm called Stochastic Gradient Descent (SGD). Its technique originates from Gradient Descent, which finds the local minimum of a differentiable function by reducing the gradient with the loss function over time. SGD reduces the computation of the standard gradient descent, instead of using the actual gradient from the full dataset it finds an estimate with a random subset of the data and optimises. SGD also overcomes the problem of being trapped at local minima. The issues with SGD are its high variance and missing global minimum when training.

Momentum attempts to reduce the high variance of SGD by accelerating the convergence the correct direction and reducing the fluctuation to wrong directions. There is an extra hyperparameter to be calculated with this method.

Adagrad, or Adaptive gradient, is an optimizer which changes its learning rate based on the frequency of parameters. Substantial changes for rare features and smaller updates for frequent features. It works exceptionally well with sparse data and can outperform SGD on robustness, but struggles in non-convex optimization tasks.

RMSprop divides gradient of the model by the square-root of the average of the square of the gradients. It can be useful for an RNN but for KGEs can impede the search in the direction of oscillations when momentum speeds up the path of the search. It operates similar to Adagrad through its squared calculating of each step which changes during training.

Adam is an optimizer based on the adaptive estimation of first-order and second-order moments to find individual learning rates for each parameter. It is essentially a mix between RMSprop and Stochastic Gradient Descent with momentum as Adam calculates scales the learning rate with squared gradients like RMSprop, and Adam uses a moving average gradient to find momentum like SGD. This method converges fast and removes the vanishing gradient and high variance. It is arguably the best available optimiser because of its efficiency for ML, but sometimes may not converge to the optimal solution and a gradient-based approach would work better.

The variable ‘k’ is the number of embedded vectors created by the model, which determines the runtime of training and its quality. If the embedding is too large, there is no extra information gained, whereas if it is too small, the semantics of the dataset is not well represented. Other than the above examples, there are a few more algorithm hyperparameters for each model that affect the training speed and calibre. The ‘ETA’ is the number of negatives embeddings created during training for each positive. An epoch is the number of training iterations through the dataset in the learning process. Early-stopping is used to monitor each epoch for loss rate as a form of regularization. If early-stopping determines increased loss, then it will end the training early to avoid overfitting. ‘Batching’ of the dataset into groups is used to speed up training. It is set by the hyperparameter Batch-size that defines the number of samples used for each epoch. Finally, ‘seed’ is used for the initialisation state by the internal pseudo-random number generator. If set to the same number, the output will give the same results. There are some other specialised hyperparameters which can be selected depending on the model which we will look at later.

4.3.3 Evaluation

To test a models performance, Ampligraph provides the function “evaluate_performance”. It follows the same procedure as seen in the paper for translational embeddings, Bordes et al. [16]. It first generates false triples by corrupting the object and the subject.

Nickel et al. [6] discuss how to artificially generate these false triples using Local Closed World Assumption (LCWA). LCWA is the strategy used to create these corrupt triples by assuming the knowledge graph is only locally complete, i.e. all real triples are also known to be true implying if a triple is not known to be true, it is false. Fundamentally, either the subject or object is corrupted at a time.

The function then removes the positive triples from the training and test sets. Finally, it ranks each test triple compared to all false triples returned. With these ranking, it is possible to compute some ranking metrics to evaluate the performance of the knowledge graph embedding. If two rankings are tied, the positive triple receives the worst rank. There are four available ranking metrics included in the Ampligraph library.

Mean Rank – This metric computes the average of a vector of rankings. It is not robust to outliers which give inaccurate results. Therefore it should be used in conjunction with the MRR, which is more reliable. Its formula is given as the following, where Q is a set of triples:

$$MR = \frac{1}{|Q|}\sum_{i=1}^{|Q|} rank(subject, predicate, object)i$$

Mean Reciprocal Rank – Computes the average of the multiplicative inverse of elements of a vector of rankings. It outputs a number between 0-1, 0 if there are no positive proposed results, and one if the positives are all ranked first.

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|} \frac{1}{rank(subject, predicate, object)i}$$

Rank of a Triple – This reports the position of a positive element against a list of negatives. It is useful for comparing predicted triples against the actual positive values.

Hits @ N - Calculates how many elements of a vector of rankings are in the top n positions. The higher the N value, the higher the score until it reaches one, which the score returns, if the actual values were all ranked in the top n positions.

$$Hits@N = \sum_{i=1}^{|Q|} 1 \; if \; rank(subject, predicate, object)i \leq N$$

There is one final evaluation tool which could prove to be the most useful for comparing embedded models called "select_best_model_ranking", implemented using a grid search or random search to compare the results of different hyperparameters for a specific model. The grid search option explores all combinations of hyperparameters, whereas the user defines a list of some parameters for the less computational random search. This function outputs the best settings for the model as a dictionary which can reduce the hyperparameter tuning time when creating a model.

4.3.4 Initial Test

A dummy dataset was made to confirm that it operates correctly with Ampligraph’s models, and to understand the construction of these models using this library. From the “full_dataset.csv” file 6,000 RDF triples were added to a new test CSV file and uploaded to Google Colab. Ampligraph’s function to load CSV files converts the CSV to a Numpy array of triples, skipping the header row which contains the title. NumPy is a Python library adding support for large, multi-dimensional arrays and matrices, and mathematical functions to operate on these. Ampligraph also has a unique function for splitting the dataset, “train_test_split_no_unseen” which carves out a test set that contains only entities and relations which also occur in the training set. It includes a Boolean parameter which allows duplications of triples in the test set when turned on. This parameter was set to false for this small test to avoid biases, and the split ratio was set to 80:20, for training to test respectively.

A ComplEx model was created with default hyperparameters, 200 epochs, batch size of 50, and with an embedding size of 150. The SGD optimiser was used at a learning rate of 0.001. The loss function used was a multiclass NLL which is standard for this model which there are more than two categories of relations. This model was then fitted using the train data with no early stopping, as it was a small dataset. The learning time was 5 minutes, returning a low loss rate of 0.002379. The trained model was then evaluated with the test data to measure its performance. The mean rank, MRR, and hits@ score were calculated for the test set.

Model	**Mean Rank**	**MRR**	**Hits @ 1**	**Hits @3**	**Hits @ 10**
ComplEx	33	0.61	0.45	0.55	0.7

Table 1 Dummy Dataset Evaluation Metrics

All of the metrics reported show high scores. Due mainly to the small size of the dataset, which means there are fewer triples to rank. 70% of real triples rank in the top 3, the Mean Rank shows an average of 33, but this may have been affected by outliers as MRR reports a very high score of 0.61. As the test model performed well on the dummy dataset and no problems occurred while training, it is time to train the real model.

4.4 Testing

There are two subsections in the testing phase of the knowledge graph embeddings, namely, Embedded Models and the Final Model. The Embedded Models section compares various models on different hyperparameters to evaluate, which performs the best with the dataset. The 2nd subsection discusses the results and selects the best performing model. The models will be assessed with standard metrics for comparison to determine to optimal knowledge graph embedding for KG completion and explainability testing.

4.4.1 Embedded Models

Four models which maybe be suitable to the dataset are the TransE, ComplEx, DistMult, and ConvE knowledge graph embeddings. All of which are state-of-the-art models, that have performed to a high prediction accuracy on large datasets, although their approaches for calculating the embedded vectors differ substantially. Ampligraph provides functions to create these models easily and to evaluate their performance. For each model, their description, setup and results are reported over the following pages.

Default Setup:

Load the “dataset.csv” file from the Google Drive, create an array of unique entities and relations (138,309 and 10 respectively), shuffle the data, and split the dataset into training and test with a 90:10 ratio. A further split from the training data, into the train and validation sets, is implemented, at a ratio of 88.8:11.1. This split makes the validation and test sets both of size 234,239 triples (10% of the full datset.csv each). The validation set is needed to use early stopping while the model is fitted to the dataset. Every 50 epochs the validation data checks for a change in MRR, to avoid overtraining the KGE. The test embedded vectors should appear in the training set to allow for an accurate evaluation of the embedded model. To guarantee this, a short piece of code checks to make sure all entities and relations in the test and validation sets occur in the training set. Any triples which contain unseen elements are excluded from the test and validation datasets. These unseen triples equate to around 10-20 for each dataset, but it is essential as otherwise, they can affect performance.

A rule was created to improve the interconnectivity and reduce the sparsity of the knowledge graph, which ensures all entities appear at least five times across the dataset. The original tests of the model showed poor results from testing because there were over 400,000 unique entities which only occurred once across the KG. Section 3.3 describes the implementation of this change to the KG to improve the performance of the embedding. Each model was tested on Google Colab’s GPU, which boasts between 10-15GB of RAM, to assist with large datasets.

The training of each model ran between 100-4,000 epochs with early stopping based on the validation sets MRR score. After an initial burn-in phase of 100 epochs, the validation set was tested every 50 epochs and stopped if the MRR rate didn’t improve for three consecutive tests.

For evaluation, positive triples were corrupted for the Object and then ranked the positive triples compared to their corruptions. Mean Rank, MRR, and Hits@ 1, 3, and 10 were recorded for each. The models which scored highest were early stopped during training, as there was no increase in rank from the validation set.

Model 1: TransE

The Translational Embeddings model finds the similarity between the subject and object using regularization techniques. The relation is a translation vector, whereby entities directly connected by relation have a short embedding distance. The TransE model only maps direct connections, whereas other distance models such as TransR and TransD can map indirectly. At the same time, this means a reduction in parameters and therefore an increased efficiency, but the model might not be complex enough to map the knowledge graph accurately.

To determine the optimal results with this model, we must consider Bordes et al. [16] evaluations. They use three benchmark datasets to test the TransE model, one WordNet [46] example, a Freebase [47] instance with 15k entities, and a Freebase [47] example with one million entities (which is most similar to our COVID dataset). Each model was fitted with several hyperparameter settings to determine the optimal predictions for the datasets. The learning rates were between {1×10^{-3} , 0.01, 0.1}, K embeddings of {20, 50}, max-margin loss of {1, 2, 10}, and either L1 or L2 for regularizers. Each dataset was trained for 1,000 epochs with early-stopping from a validation set. The optimal parameters for each dataset are, with their results displayed on the table 2 below:

WordNet: k = 20, ʎ = 0.01, Margin = 2, and Regularizer = L1

FB15K: k = 50, ʎ = 0.01, Margin = 1, and Regularizer = L1

FB1M: k = 50, ʎ = 0.01, Margin = 1, and Regularizer = L2

	WordNet		FB15k		FB1M	
Metric	Mean Rank	Hits@10	Mean Rank	Hits@10	Mean Rank	Hits@10
TransE	263	75.4	243	34.9	14,615	34

Table 2 Optimal TransE Results on Benchmark Datasets [16]

Batching is needed to train this embedding for our large dataset, which reduces the number of triples trained per epoch, denoted by the batch-size hyperparameter. Batch-sizes of {64, 100, 128} were tested.

A variety of learning rates were tested dependant on the type of optimizer used {0.1, 0.01, 1×10^{-3}, 5×10^{-4}}. The optimizers SGD, Adagrad and Adam all produced acceptable results, but Momentum didn’t, possibly because of the optimisation isn’t convex. Adagrad works best with a high learning rate which adapts overtime whereas SGD and Adam are most efficient with learning rates less than 1×10^{-3}.

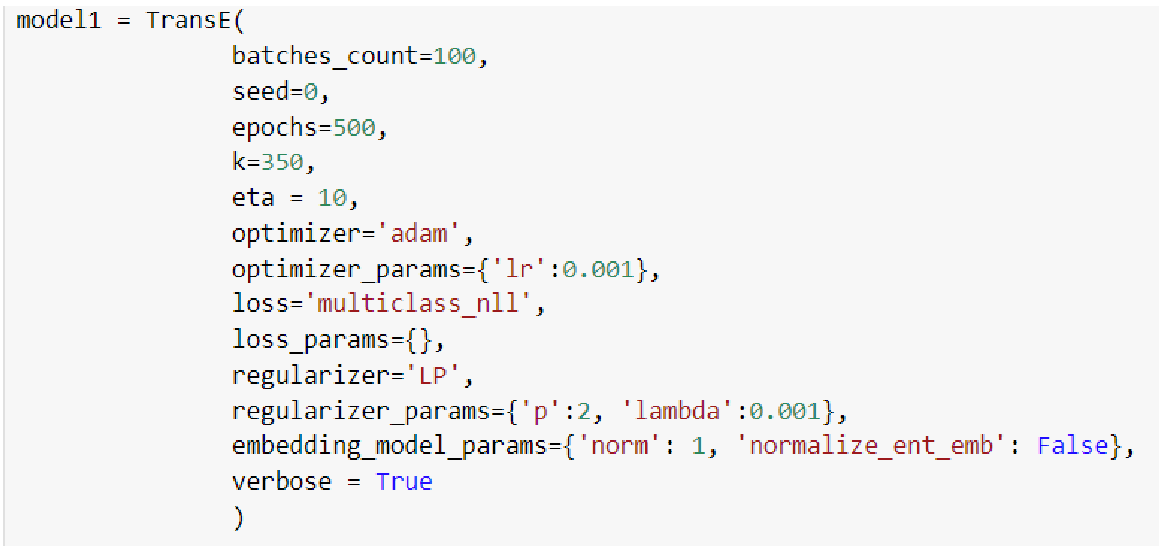
Tests with an ETA of ranges {1, 2, 5, 10, 20} were used. ETA is the number of negatives generated in training for each positive as a test metric. Higher values are more inefficient and cause training time to increase exponentially. Therefore it is more beneficial to keep this low and increase the batch-size training the model.

Embedding space dimensionality, K value, ranged from {20, 50, 100, 150, 200, 250, and 350}. Increasing this value has a major effect on training times and the ranking scores plateau after around 200.

Loss functions of pairwise, NLL, self adversarial and Multiclass NLL were tested. For pairwise max marginal loss function, we were using tests of margins with {0.5, 1, and 10}. Multiclass NLL was the most effective in producing the highest MRR scores.

Both L1 and L2 were tested as regularizers with a range of lambda values: {0.01, 0.001, $1\times 10^{-3}, 1 \times 10^{-5}$}. L2 regularizer with a lambda of 0.001 worked best.

Figure 16 shows the setup in Ampligraph of one of the test models. This figure depicts how the model is made, and its hyperparameters are set.



```
model1 = TransE(
                    batches_count=100,
                    seed=0,
                    epochs=500,
                    k=350,
                    eta = 10,
                    optimizer='adam',
                    optimizer_params={'lr':0.001},
                    loss='multiclass_nll',
                    loss_params={},
                    regularizer='LP',
                    regularizer_params={'p':2, 'lambda':0.001},
                    embedding_model_params={'norm': 1, 'normalize_ent_emb': False},
                    verbose = True
                    )
```

Figure 16 Example TransE Model Parameter Setting

Table 3 displays the results of the models which scored the highest in the evaluation.

Their optimal hyperparameters tested were:

1. k = 100, optimizer = SGD, Batch-size = 128, ETA = 2, Pairwise Loss, margin = 5.
2. k = 100, optimizer = Adagrad, Batch-size = 64, ETA = 2, Pairwise Loss, margin = 2.
3. k = 150, optimizer = Adagrad, Batch-size = 64, ETA = 5, Pairwise Loss, margin = 1.
4. k = 150, optimizer = Adagrad, Batch-size = 64, ETA = 5, Multiclass NLL, Reg = L2, ʎ=0.01.
5. k = 200, optimizer = Adagrad, Batch-size = 64, ETA = 10, Multiclass NLL, Reg = L2, ʎ=0.01.
6. k = 150, optimizer = Adagrad, Batch-size = 64, ETA = 5, Multiclass NLL, Reg = L2, ʎ=0.01.
7. k = 350, optimizer = Adam, Batch-size = 100, ETA = 10, Multiclass NLL, Reg = L2, ʎ=0.001.

TransE	Mean Rank	Mean Rank Reciprocal	Hits@1	Hits@3	Hits@10
Test 1	4,536.66	0.06	0.03	0.06	0.10
Test 2	5,228.71	0.07	0.04	0.07	0.11
Test 3	7,546.51	0.08	0.04	0.08	0.13
Test 4	3,082.61	0.09	0.05	0.09	0.15
Test 5	3,435.71	0.09	0.05	0.10	0.17
Test 6	3,131.15	0.10	0.07	0.12	0.19
Test 7	3609.37	0.14	0.09	0.15	0.22

Table 3 TransE Model Metrics

Model 2: DistMult

DistMult is a model built under a connected learning framework, where the entities are embedded vectors trained on a neural network, and relations are mapped with bilinear links. It shows an improvement in metrics, compared to the TransE model, for benchmark datasets.

Yang et al. [36] describe the DistMult model and evaluate it with different hyperparameters settings to find the optimal. Three datasets were tested, two of which were also tested on TransE: WordNet [46] and Freebase [47] 15K entity, and the third is an enhanced Freebase [47] 15k with only the most frequent relations. The models optimised on both Adagrad and SGD using mini-batching, with a batch-size between {10, 50, 100}. Two ETA negatives were generated for each epoch. L1 and L2 regularization (lambda=1×10^{-3}) were both tested, with a dimensionality of K=100. The Freebase [47] models ran for 100 epochs, and the WordNet [46] dataset ran for 300, all three with learning rates between {0.1, 0.01, 0.001}.The optimal parameters for each dataset are, with their results displayed on table 4 below:

WordNet: Epochs = 300, k = 100, ETA = 2, Pairwise loss, Margin = 1

FB15K: Epochs = 100, k = 100, ʎ = 0.001, Regularizer = L2, ETA = 2

FB15k-401: Epochs = 100, k = 100, ʎ = 0.001, Regularizer = L2, ETA = 2

	WordNet		FB15k		FB15k-401	
Metric	MRR	Hits@10	MRR	Hits@10	MRR	Hits@10
DistMult	0.8	94.2	0.3	57.7	0.4	58.5

Table 4 Optimal DistMult Results on Benchmark Datasets [36]

Batching of the dataset is implemented for this model also with batch-sizes {10, 16, 32, 64} tested.

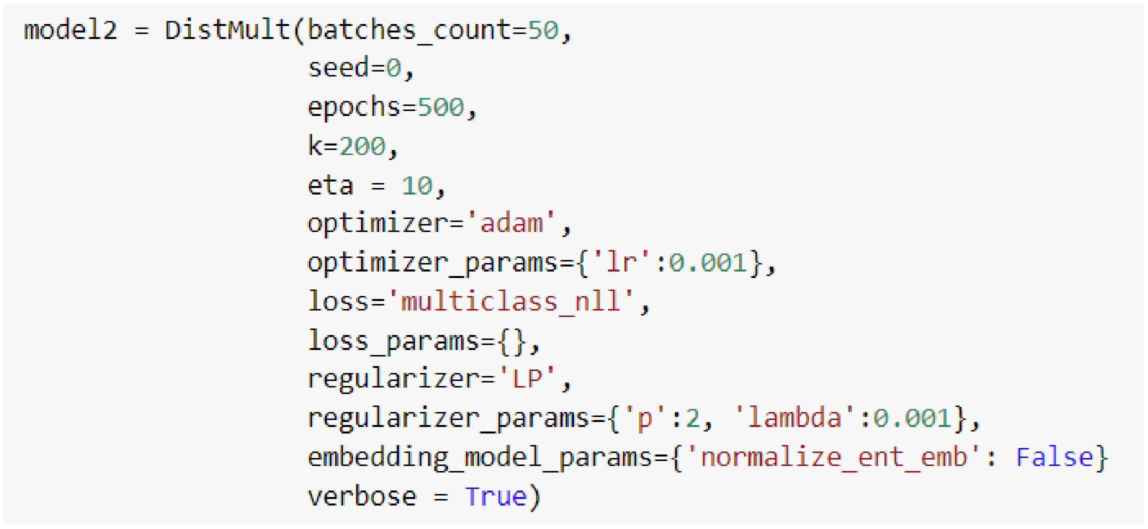
The optimizers SGD, Adagrad and Adam all used different learning rates to accommodate for their operation. Learning rates of {0.1, 0.01, 0.001, 0.0001} were implemented.

Tests with an ETA of ranges {2, 5, 10} were used.

K values between {50, 100, 150, 200, 350} were tested.

Loss functions of pairwise, NLL, self adversarial and Multiclass NLL were tested. For pairwise max marginal loss function, we were using tests of margins with {0.5, 1, 2, 5, and 10}. Multiclass NLL was the most effective in producing the highest MRR scores.

L2 regularizer with a lambda of 0.001 was the optimal but lambda values of {0.01, 0.001, $1\times 10^{-3}, 1 \times 10^{-5}$} were all tested.



```
model2 = DistMult(batches_count=50,
                  seed=0,
                  epochs=500,
                  k=200,
                  eta = 10,
                  optimizer='adam',
                  optimizer_params={'lr':0.001},
                  loss='multiclass_nll',
                  loss_params={},
                  regularizer='LP',
                  regularizer_params={'p':2, 'lambda':0.001},
                  embedding_model_params={'normalize_ent_emb': False}
                  verbose = True)
```

Figure 17 Example of DistMult Model Parameter Settings

Table 5 displays the results of the best seven models.

Their optimal hyperparameters tested were:

1. k = 100, optimizer = SGD, Batch-size = 10, ETA = 2, Pairwise Loss, margin = 1.
2. k = 150, optimizer = Adagrad, Batch-size = 16, ETA = 2, Pairwise Loss, margin = 2.
3. k = 150, optimizer = Adagrad, Batch-size = 16, ETA = 5, Pairwise Loss, margin = 5.
4. k = 150, optimizer = Adagrad, Batch-size = 32, ETA = 10, NLL, Reg = L1, Λ=0.01.
5. k = 200, optimizer = Adagrad, Batch-size = 64, ETA = 5, NLL, Reg = L2, Λ=0.01.
6. k = 350, optimizer = Adagrad, Batch-size = 64, ETA = 5, Multiclass NLL, Reg = L2, Λ=0.001.
7. k = 350, optimizer = Adam, Batch-size = 50, ETA = 10, Multiclass NLL, Reg = L2, Λ=0.0001.

DistMult	Mean Rank	Mean Rank Reciprocal	Hits@1	Hits@3	Hits@10
Test 1	5,503.81	0.04	0.02	0.04	0.07
Test 2	4,700.57	0.05	0.03	0.05	0.09
Test 3	3,450.56	0.05	0.03	0.06	0.10
Test 4	2,720.17	0.07	0.04	0.07	0.13
Test 5	2,379.26	0.10	0.06	0.12	0.16
Test 6	7,546.41	0.17	0.09	0.14	0.19
Test 7	2,477.12	0.17	0.13	0.19	0.24

Table 5 DistMult Model Metrics

Model 3: ComplEx

The ComplEx model maps entities and relations into a complex vector representation space. They are created using latent factorization, calculated using the Hermitian dot product. This model outperforms both the TransE and DistMult models on large benchmark datasets for standard evaluation metrics, making it arguably the best approach for this project.

Trouillon et al. [42] test the results of ComplEx, which must be consulted to help perfect our testing. Freebase [47] 15K and WordNet18 [46] datasets were used as the benchmark datasets to compare ComplEx with several other models. They were evaluated on the MRR and Hits@ rate of 1, 3, and 10. Each model ran for 1000 epochs, but early-stopping was used during training validated with MRR with tests every 50 epochs. A large range of tests was performed for different hyperparameters. K was set for a value between {10, 20, 50, 100, 150, 200}, lambda regularization was set between {0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0}, ETA was set between {1, 2, 5, 10}, learning rates of {1.0, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01} were tested. The batch size used was 100, with optimal results found when K=150. L2 was shown to be most effective with 0.1 optimal lambda value, and ETA of 10 producing the best results depending on the dataset. Adagrad optimised the training, which changes the learning rate over time based on the loss function. These tests showed setting the initial learning rate to a higher value produced optimal operation.

The optimal parameters for each dataset were, with their results displayed on table 6 below:

WordNet18: Epochs = 1000, k = 150, ETA = 10, Pairwise loss, Margin = 1

FB15K: Epochs = 1000, k = 150, ETA = 10, ʎ = 0.1, Regularizer = L2

	WN18				FB15K			
Metric	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
ComplEx	0.587	0.936	0.945	0.947	0.242	0.599	0.759	0.84

Table 6 Optimal ComplEx Results on Benchmark Datasets [42]

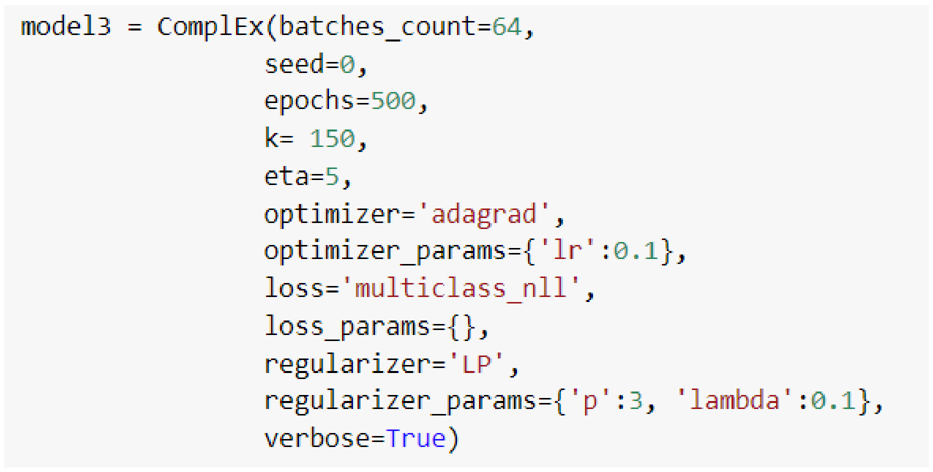
The negative log-likelihood loss function is most suited to the ComplEx model as it contains more parameters. For our dataset, as there are multiple relations multiclass NLL was most effective for testing.

Regularization enhances NLL, and the model was trained with both L1 and L2. A third option was to add Canonical Decomposition to ComplEX, converting it to the ComplEx-N3 model [66]. This change provided an improvement on the L2 regularizer.

K embedding vectors of {50, 100, 150, 200, 350, 400} were tested.

ETA negative triples of {2, 5, 10, 20} per epoch were also computed, a median value was optimal.

Optimizers Adam, Adagrad, and SGD all proved useful, but Adagrad produced all the optimal results.



```
model3 = ComplEx(batches_count=64,
                 seed=0,
                 epochs=500,
                 k= 150,
                 eta=5,
                 optimizer='adagrad',
                 optimizer_params={'lr':0.1},
                 loss='multiclass_nll',
                 loss_params={},
                 regularizer='LP',
                 regularizer_params={'p':3, 'lambda':0.1},
                 verbose=True)
```

Figure 18 ComplEx Model Creation Code

Table 7 displays the results of the top seven trained models.

Their optimal hyperparameters tested were:

1. k = 100, Optimizer = SGD, Batch-size = 64, ETA = 5, NLL, Reg = L2, Λ=0.1.
2. k = 100, Optimizer = Adagrad, Batch-size = 64, ETA = 5, Multiclass NLL, Reg = L1, Λ=0.01.
3. k = 100, Optimizer = Adagrad, Batch-size = 64, ETA = 5, Multiclass NLL, Reg = L2, Λ=0.01.
4. k = 150, Optimizer = Adagrad, Batch-size = 64, ETA = 5, Multiclass NLL, Reg = L2, Λ=0.001.
5. k = 150, Optimizer = Adagrad, Batch-size = 64, ETA = 5, Multiclass NLL, Reg = L3, Λ=0.1.
6. k = 350, Optimizer = Adam, Batch-size = 100, ETA = 5, Multiclass NLL, Reg = L3, Λ=0.001.
7. k = 400, Optimizer = Adam, Batch-size = 100, ETA = 5, Multiclass NLL, Reg = L3, Λ=0.001.

ComplEx	Mean Rank	Mean Rank Reciprocal	Hits@1	Hits@3	Hits@10
Test 1	7,902.20	0.10	0.03	0.05	0.10
Test 2	4,343.83	0.10	0.05	0.08	0.13
Test 3	4354.85	0.12	0.05	0.09	0.14
Test 4	4834.21	0.13	0.06	0.11	0.15
Test 5	2308.43	0.15	0.08	0.12	0.19
Test 6	2441.38	0.20	0.14	0.18	0.25
Test 7	2104.23	0.25	0.16	0.20	0.27

Table 7 ComplEx Model Metrics

Model 4: ConvE

ConvE is a multilayer convolutional network which is parameter efficient and more expressive than other state-of-the-art models. It contains ConvE-specific hyperparameters the ability to change the structure and sizes of the layers. These include the ability to edit the number of convolution feature maps, change the kernel size, and add dropout to the embedded, convolution, or dense layers.

Dettmers et al. [44] compare the parameters and evaluation of the ConvE model, which will help to find the most optimal results for this project. The WordNet18 [46], Freebase 15k [47], and YAGO3-10 [24] were the three benchmark datasets used for testing. Grid search across multiple hyperparameters was completed with ranges of embedding dropout {0.0, 0.1, 0.2}, feature map dropout {0.0, 0.1, 0.2, 0.3}, projection layer dropout {0.0, 0.1, 0.3, 0.5}, embedding size k {100, 200, 300}, batch size {64, 128, 256}, learning rate {0.001, 0.003}, and label smoothing {0.0, 0.1, 0.2, 0.3}. The optimal results were found for the three datasets using small filters (3x3) with: embedding dropout 0.2, feature map dropout 0.2, projection layer dropout 0.3, learning rate 0.001, and label smoothing 0.1.

The optimal parameters for each dataset were, with their results displayed on table 8 below:

WordNet18: k = 300, Label Smoothing = 0.1, Optimizer = SGD

FB15K: k = 200, Label Smoothing = 0.1, Optimizer = Adagrad

YAGO3-10: k = 150, Label Smoothing = 0.1, Optimizer = Adam

	WN18				FB15k				YAGO3-10			
Metric	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
ConvE	0.943	0.935	0.946	0.956	0.657	0.558	0.723	0.831	0.44	0.35	0.49	0.62

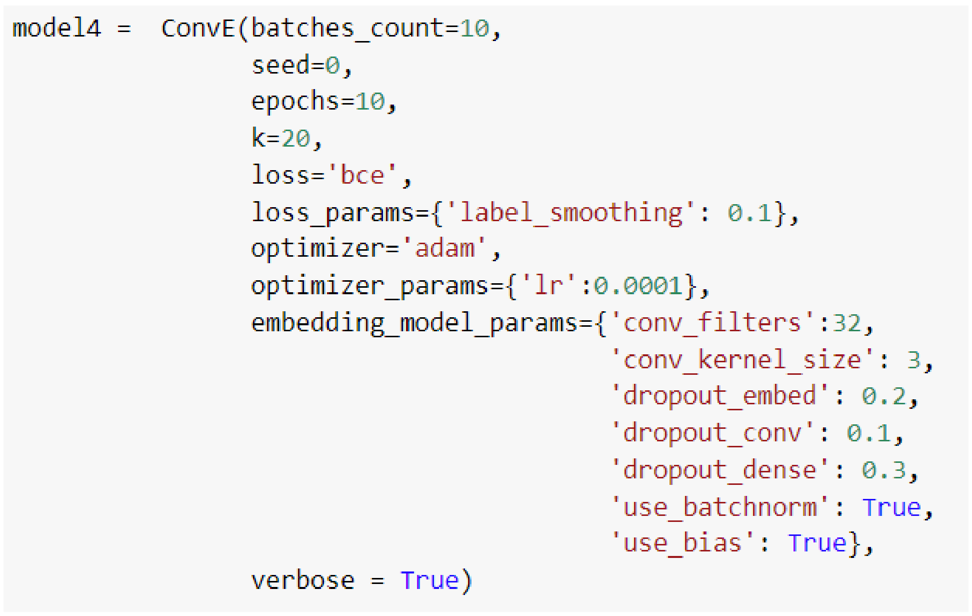
Table 8 Optimal ConvE Results on Benchmark Datasets [44]

ConvE allows the ability to edit the convolution network settings, including the filter sizes, number and dropout probabilities for each layer. These settings help create the model

The loss function of ConvE must be Binary cross-entropy (BCE) and can't be used on other models either. BCE is a logistic loss function used for binary classification problems.

Label smoothing is a hyperparameter used with BCE to apply one-hot output. One-hot output encodes the categorical labels into their embeddings in binary format, instead of using integer encoding like the other KGE models. The length of each one-hot encoded vectors is equal to the total number of unique embedded vectors. All the digits in a vector are set zero except the actual category of the embedding, which is set to one. The fraction given to the label smoothing hyperparameter assigns a higher value than zero to negative labels, and lower than one to the positive labels. For example, if label smoothing is set to 0.1, there is a smaller gradient given to an incorrectly predicted element than if rigid binary classification was used.

No ETA is used with ConvE as no negatives are needed to compute the embeddings. It learns much more like an ML algorithm than the other KGE we have tested.



```
model4 =  ConvE(batches_count=10,
                seed=0,
                epochs=10,
                k=20,
                loss='bce',
                loss_params={'label_smoothing': 0.1},
                optimizer='adam',
                optimizer_params={'lr':0.0001},
                embedding_model_params={'conv_filters':32,
                                        'conv_kernel_size': 3,
                                        'dropout_embed': 0.2,
                                        'dropout_conv': 0.1,
                                        'dropout_dense': 0.3,
                                        'use_batchnorm': True,
                                        'use_bias': True},
                verbose = True)
```

Figure 19 ConvE Model Creation Code

Figure 19 shows how ConvE is created in Ampligraph with some standard hyperparameter settings. The k, epochs, and batch-size are all set much lower than optimal as ConvE crashed Colabs CPU, and GPU while training on the dataset. At first, this was believed to be due to incorrect hyperparameter tuning, but after rigorous testing, the only model which worked was the validation dataset of 20,000 triples. Even with this, the hyperparameter settings required much lower computation than usual just to get the model to run.

Another possibility was that the ConvE model couldn't handle the dataset with over 2 million triples. The ConvE model had trained as Ampligraphs manual and paper states with many large-scale datasets, so even bigger than CKG. These tests ran with slightly more GPU space, but Colabs 15GB GPU should have been enough to train at least a basic model.

The most likely explanation of this issue is the large number of unique entities in the dataset (138,000). One-hot encoding requires massive storage space to embed all these unique entities which would all be of length 138,000. Increasing the filter size to 10 showed no improvement even though there was half the number of unique entities. Increasing this further would lead to a skewed version of the dataset, which would represent the COVID-19 data in a biased way. For this reason, no more ConvE models were trained.

4.4.2 Final Model

The “select_best_model_ranking” function required a lot of computation as training several models with different hyperparameters took over the 12 hour Google Colab time limit. For this reason, it wasn’t implemented, and each model was trained individually. Each model performed to an adequate level apart from ConvE, which no results were obtained. This was unfortunate as the algorithm is one of the most modern approaches and could offer an interesting comparison to the others.

Through performing these tests, some common patterns were found about how to optimise KGE models for this dataset. The Multiclass NLL loss function produced the optimal results for all models. This loss function is useful for datasets which have many relations which the CKG has 11 in total. Another pattern seen across the models was the success of the Adam optimiser, which significantly achieved more substantial results than its counterparts. Low learning rates outputted the lowest model loss for all optimisers barre Adagrad, which worked best with 0.1. The L2 Regualarizer always improved on the metrics of L1 with small lambda values showing the best efficiency. The different initialisers had minimal effect on the output as long as they weren’t zero. Xavier was the default initialiser used in testing. The combination of lower ETA values and higher K values generated excellent metric outputs without considerably increasing runtime. To this effect, it was vital to set the batch-size under 100, to reduce overall computation. Early-stopping was crucial in finding the best number of epochs to use, checking the MRR of the validation set every 50 generations. The leading models all stopped early before the specified number of epochs as the MRR rate hadn’t improved for three consecutive tests.

The TransE and DistMult achieved similar results for the different hyperparameter settings with DistMult just outperforming TransE with increased ETA and K values. The highest Hits@10 rate was 0.24 for DistMult and 0.22 for TransE with MRR of 0.17 and 0.14 respectively. TransE had more outliers producing a larger Mean Rank of 3609 compared to DistMults 2477. The training time for DistMult was also shorter even though because its method for calculating embeddings is more efficient even though they have a similar number of parameters. The ComplEx model outputs lower metrics for most of its tests compared to TransE and DistMult scoring MRR of 0.13 and Hits@10 of 0.15. It is a more challenging model to tune as it is complicated and has a much slower training time. Only when CD is added by changing the regulariser to p=3 that creates the ComplEx-N3 algorithm, which dramatically improves its results. ComplEx-N3 scores Hits@10 of 0.27 ad MRR of 0.25 at its maximum. This is the best model as it also has a low MR meaning the variance in embeddings. All future tests in chapters 5 and 6 use this model for prediction and visualisation tasks.

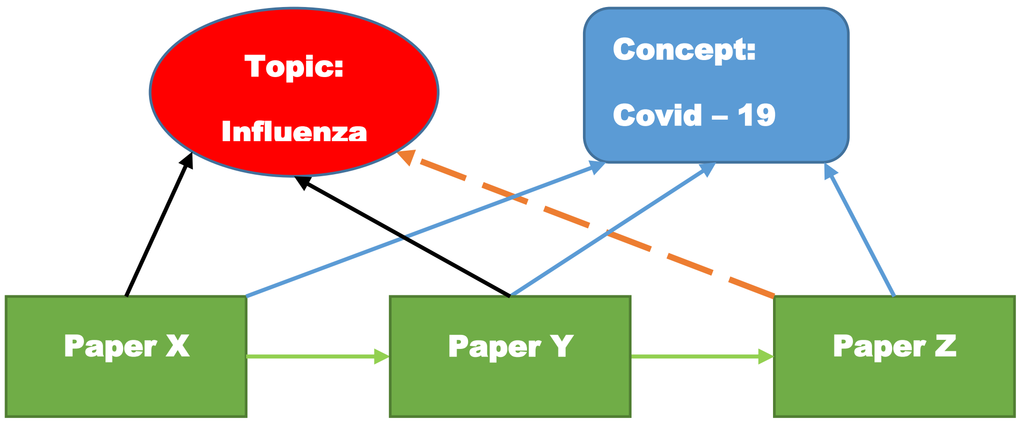
Overall the training of these models underperformed in comparison to benchmark datasets. There are many reasons why this could be the case, including the hyperparameter tuning, the dataset used, and libraries implemented. Each KGE was tested for a large variety of different hyperparameters which were directly comparable to those used in the research papers for each model. Therefore, it is unlikely that further evaluating more hyperparameters could significantly enhance performance. Ampligraph's libraries could not be at fault either as each KGE was assessed with the benchmark datasets used in research papers, obtaining very similar metric scores. Thus the reason the models struggled to accomplish scores is because of the dataset. There were many unique entities which had too few relations even after the filtration process which removed entities with less than five links. This filtration procedure did progress the dataset, but when triples were detached, it reduced the number of connections for other entities, making it harder for the KGE to learn patterns from the knowledge graph. Though the results are lower than expected, they can still effectively predict relationships and clusters for the KG making this process successful. The next two chapters will implement the best model for further applications in KGC and Explainability.

5 Link Prediction

5.1 Overview

Knowledge graph completion is the next stage of this project. There are several methods to implement KGC, such as Entity Resolution, Probabilistic Soft Logic, and Knowledge Graph Embeddings. This project focuses on KGE approaches, although a description of the others is in the literature review (section 2.5). Any KGE algorithm mentioned in chapter 4 can implement link prediction, but the best predictions come from the models with the highest-ranking score. The chapter takes the best model, ComplEx-N3 from section 4.4, to test out some KGC methods to gain an understanding of its operation and to improve the CKG quality. Section 5.2 discusses the techniques used to achieve KGC and the tools which assisted the process. Section 5.3 compares the approaches taken and establishes the benefits of undergoing this procedure.

To give a brief example of how a link prediction works, the diagram below demonstrates a knowledge graph with relevant Covid-19 data. The coloured blocks are entities from the dataset, which include papers, topics, and concepts. The arrows between entities are the relations, and their colours represent the relation type, listed in the legend. As Paper X and Paper Y are both cited in Paper Z, and all three contain the concept of Covid-19, we can predict the missing topic link of Paper Z. This prediction should improve the quality and variety of the knowledge graph, making it more useful to other tasks.



Link Type	Shape
Topic of Paper	→
Paper Cited by Paper	→
Missing Link Predicted	⇢
Concept of Paper	→

Figure 20 Link Prediction Covid-19 Example with Legend

5.2 Applying Link Prediction

5.2.1 KGC Tools

The Python library used in chapter 4, Ampligraph, also has functions to implement some KGC methods. These functions are all in Ampligraph's 'Discovery' module, which takes the KGE models and uses them to find out new information about the knowledge graph. Section 5.2.2 compares the implementation of some of these functions with KGC tests. Here is an explanation of each.

"discover_facts": This function predicts new triples for the knowledge graphs. It generates new candidate statements, using a defined strategy, and then ranks them alongside the corrupted triples to determine the most likely. It returns the highest-ranked as factual statements. There are several different strategies to formulate the new triples, including random uniform sampling, less frequent entities, entities with few relations, and smaller graph clusters. Random uniform is an exhaustive search which requires much computation. The logic of the other three strategies is to find new triples in sparse regions of the graph as they are more likely to be incomplete. The different hyperparameters include the threshold of ranking, the maximum number of triples created, and the option to target a specific relation for predictions. This method is very accurate at predicting new triples for high quality embedded models. The more links predicted correctly, the more complete the knowledge graph is.

"find_clusters": A function which implements clustering of entities in a KGE model to get a better understanding of the relationships between entities. Cluster detection occurs over the embedding space of the triples to test the quality of the KGE. The clustering algorithms are from sci-kit learn [67], which provide a variety of different options ranging from K-means clustering to DBSCAN. These algorithms perform evaluations over either entities, triples, or relations, returning an index of the cluster for each. A plot of the labelled entities in their respective groups can allow data visualisation of the model. This clustering function portrays a broader depiction of the knowledge graph, aiding in the understanding of patterns in the interconnected network.

"find_duplicates": Determines elements in a triple which are the same, but have different names, based on their embedded relationships in the model. It can evaluate predicate, entities and triples of a model, in search of duplicates. Duplicates are points which are less than a defined threshold in the KGE. This threshold can be manually defined or set by an optimising algorithm. A high MRR score is required for this function to work effectively. The function outputs the threshold used and lists of duplicate entities. The KG can merge found duplicates to reduce the number of unique elements making searching and runtime computation more efficient.

"query_topn": Predicts the most likely missing element of a triple when given two components for completion. It outputs a ranking of the probable unknown element of the triple. It can return already known positive statements along with newly predicted ones and therefore is not solely a KGC method. The prediction of each triple, made individually, is ineffective for large-scale improvements, especially when expected completions may be already known.

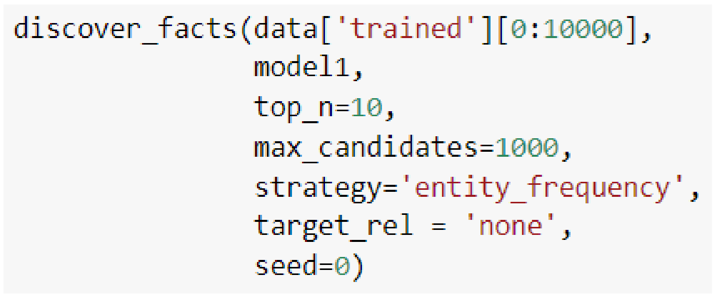
5.2.2 Testing

The KGE chosen is the ComplEx-N3 model which performed best in the previous chapter, decided in section 4.4.2. Three tests were completed using Ampligraphs modules "discover_facts", "find_clusters", and "find_duplicates". These offer completion to our knowledge graph in different ways, and therefore it is useful to test all. The "query_topn" method was too individualistic and couldn't improve the overall knowledge graph. Consequently, no tests were performed with it.

<u>Module 1: Discover Facts</u>

Initial testing of the "discover_facts" module was implemented to understand its operation, as shown in Figure 21. The trained model and a small subset of the training data was inputted to this module. The ranking threshold was set to the top 10 for newly predicted triples to be considered accurate. The first strategy tested to find new triples was the frequency of entities in the knowledge graph. This strategy generates a maximum of 1000 new candidates which are ranked in comparison with the actual triples.

This test outputted 17 predicted facts about the knowledge graph which were not already present. These are found for entities which have fewer connections, and therefore may be missing some relations. The target relations was not set, which lead to a much slower runtime as all link types were predicted. The entity frequency strategy worked effectively, but a clustering strategy may outperform it.



```
discover_facts(data['trained'][0:10000],
               model1,
               top_n=10,
               max_candidates=1000,
               strategy='entity_frequency',
               target_rel = 'none',
               seed=0)
```

Figure 21 An example creation of the "discover_facts" module

The next phase was to test the full training dataset. This stage required much more computation, but the predictions represent the data more effectively. Many of the relations should be predicted as they are facts about the dataset, for example, titles of unique IDs, authors of papers, authors affiliated with institutes, and direct citations of research papers. With these removed there are only two remaining relations to predict, namely "associated_concept" and "similarity". New predictions from "associated_concept" can find relationships between concepts in research papers. Associations found with the "similarity" relation, deduce connections between research papers which are alike. Again, candidates that rank in the top 10 are deemed to be true. Three strategies were tested, but only entity frequency produced substantial results. A total of 1420 new predictions were inferred with this test, 940 "associated_concept" and 480 "similarity" relations. An example of one of these triples is shown in the plot below in Figure 22 and Figure 23. The NetworkX library was used to visualise this example of the knowledge graph, as seen in section 3.5.

The first image (Figure 22) displays papers A and B and three different concepts, namely infection, vaccine, and respiratory complaint. Paper A only links to the infection and vaccine entities, whereas Paper B links to all three. One of the predicted triples found with the "discover_facts" module was:

Subject:	db81bece-e5f3-45f0-9044-3085afc4c739	(Paper A)
Relation:	associated_concept	
Object:	6197fea5-26eb-4c10-aee3-4d772d29fd8f	(respiratory complaint)

Figure 23 displays the latest link added to the knowledge graph, which gives completion to this paper entity.

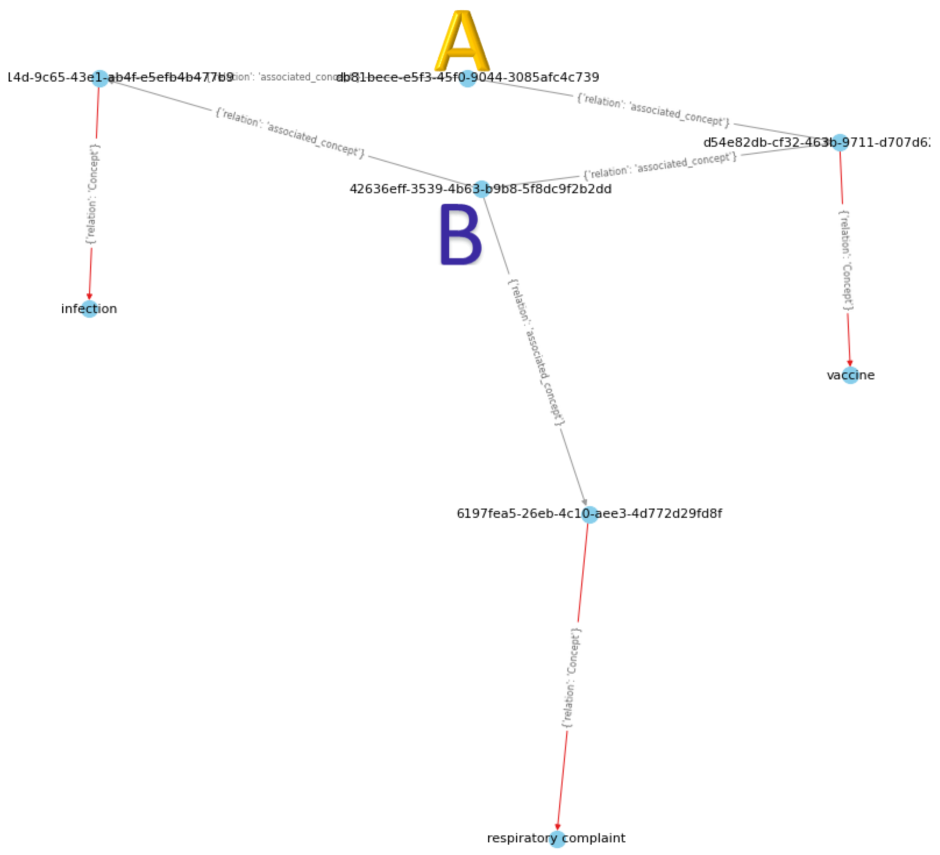

A
B
l4d-9c65-43e1-ab4f-e5efb4b477b9
db81bece-e5f3-45f0-9044-3085afc4c739
d54e82db-cf32-463b-9711-d707d6
42636eff-3539-4b63-b9b8-5f8dc9f2b2dd
6197fea5-26eb-4c10-aee3-4d772d29fd8f
infection
vaccine
respiratory complaint
{'relation': 'associated_concept'}
{'relation': 'Concept'}

Figure 22 Link Prediction example of an "associated_concept" plotted with NetworkX

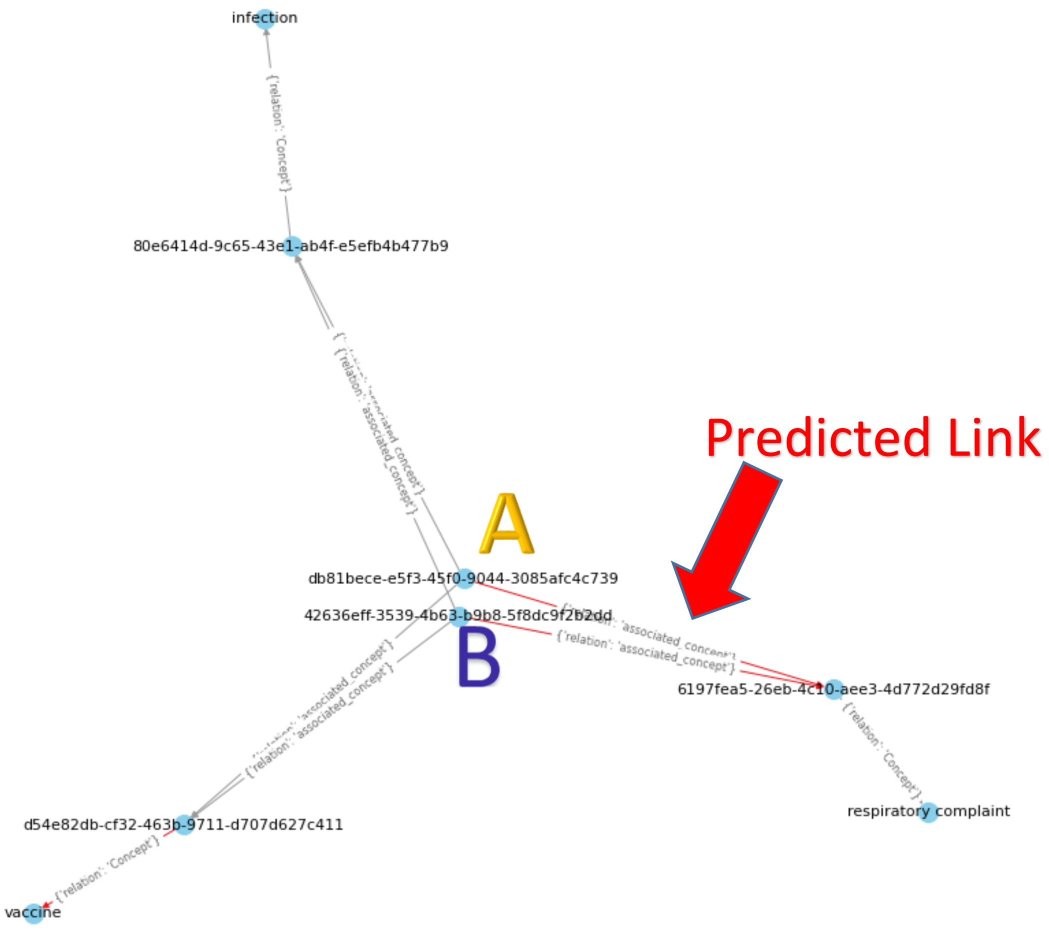
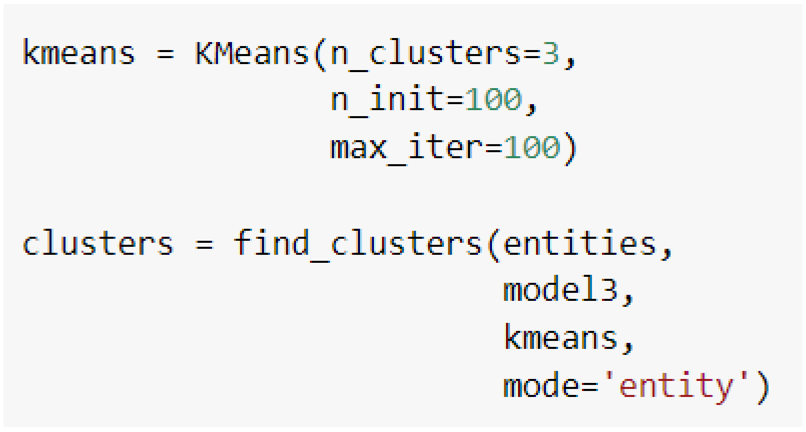

Predicted Link
A
B
infection
80e6414d-9c65-43e1-ab4f-e5efb4b477b9
db81bece-e5f3-45f0-9044-3085afc4c739
42636eff-3539-4b63-b9b8-5f8dc9f2b2dd
6197fea5-26eb-4c10-aee3-4d772d29fd8f
d54e82db-cf32-463b-9711-d707d627c411
respiratory complaint
vaccine
{'relation': 'associated_concept'}
{'relation': 'Concept'}

Figure 23 Paper A with new "associated_concept" link predicted

The entity frequency strategy used finds this new connection as both paper entities have few total relations and share common concepts. The affiliation between these papers and concepts is logical in the real-world, which shows that the knowledge graph represents the dataset effectively. All the other predicted links which ranked in the top ten most probable were also added for knowledge graph completion. This KGC infers more information about the dataset giving the KG more meaning and improving the accuracy of the KGE. This module is a productive method for completing knowledge graphs but also to enhance the interpretation of the network.

Module 2: Find Clusters

The “find_cluster” module allows each element to labelled into a specific cluster to perform analysis between natural and predicted groupings of entities. For this initial test, shown in Figure 22, the collections were found between entities in the dataset by setting the mode to ‘entity’. This mode groups unique entities and it requires less computation than clustering individual triples.



```
kmeans = KMeans(n_clusters=3,
                n_init=100,
                max_iter=100)

clusters = find_clusters(entities,
                         model3,
                         kmeans,
                         mode='entity')
```

Figure 24 An example creation of the "find_clusters" module

The K-means clustering algorithm was used to find these groups built with the sci-kit learn library. The number of clusters is set to 3 with an initial distance of 100 between their centroids. This algorithm will always converge with enough iterations, but this might be a local minimum. An initial space is set between centroids to avoid this making it preferable compared to random initialisation. There is a maximum of 100 iterations to find the optimal positions of the centroid for each cluster. K-means works best with a low number of sets and minimises within-cluster sum-of-squares using the following formula:

$$\sum_{i=0}^{n} \min_{\mu j \in C} (\|xi - \mu j\|^2)$$

The formula minimises the inertia, the distance between entities and the centroid for all the embeddings in that cluster. This inertia is not normalised, but values closest to zero are optimal. The problem with this metric is the increased dimensionality for larger embedding sizes. To reduce the dimensionality Principal Component Analysis (PCA) is performed, which is necessary to plot the embedding also. PCA uses Singular Value Decomposition (SVD) to project the embedded vectors of each entity to the first principal components. These components are the direction that maximizes the variance of the projected data. For the KGE model, there are 400 embeddings for each entity. After PCA, there are only two embeddings for each entity.

Each entity can now be plotted on a two-dimensional graph using the Matplotlib library. This library allows the creation of a scatter plot with the two embeddings as the X and Y axes. Its cluster defines the colour of each entity. Figure 25 depicts the three groups with the dataset.

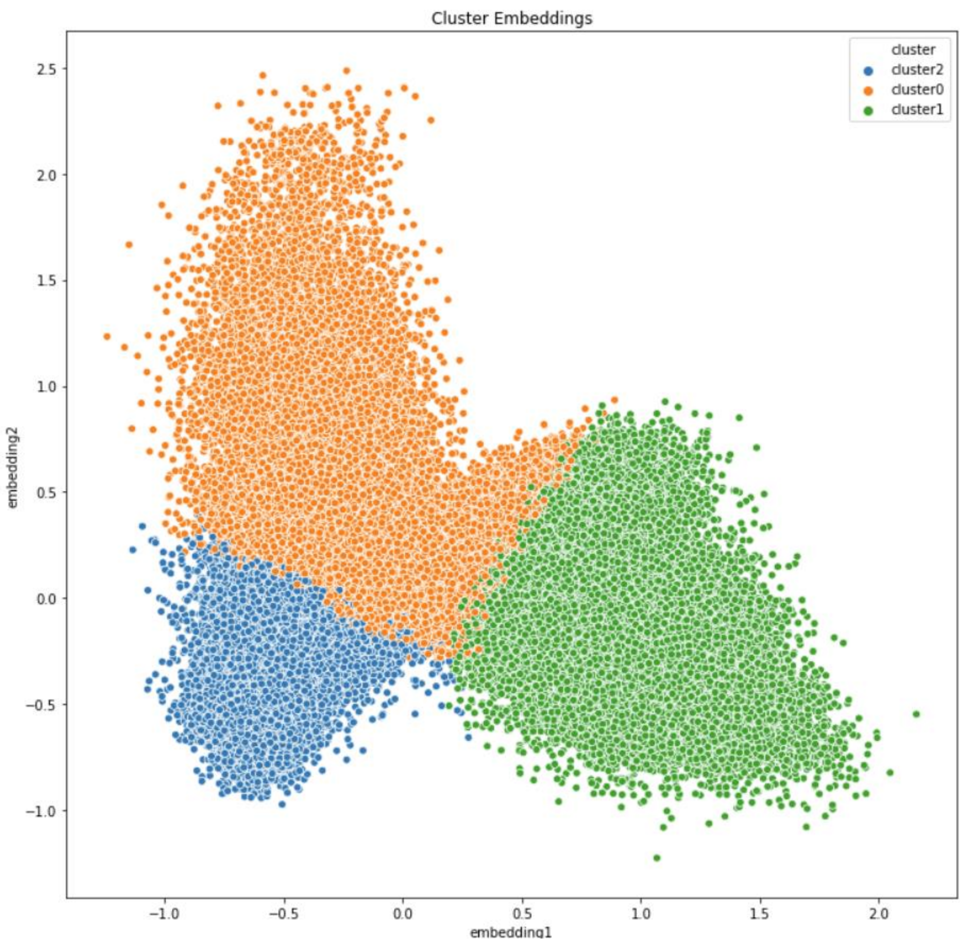


Figure 25 K-means Clustering of size three from entities in the KG

Three is the default number of clusters used with K-means, but it might not be the best. An evaluation method is necessary to optimise the number of groups to use. The most common way to implement this is by using the Elbow Method. This method calculates the sum of the squared distances from each point to its centroid, for a range of different cluster values. The sum of the squares was calculated between K values in the range 1-12. The change in slope determines the optimal number of clusters. Figure 26 graphs the results, highlighting the optimal (K = 9) with the white vertical dotted line.

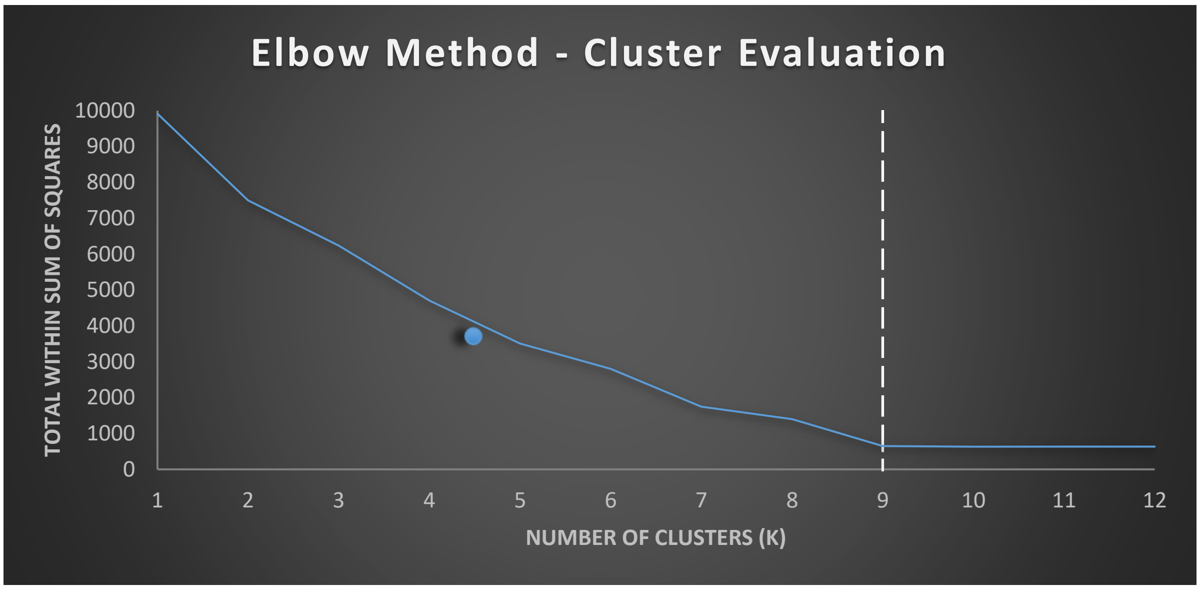


Figure 26 Elbow Method to determine optimal clusters

The slope of the graph begins to level when K is nine and stays consistent for the other three tests after meaning this should be the best number of K dimensions to choose. The most effective way to compare the quality of this task is by comparing the predicted clusters to the actual groupings present in the dataset. The natural groups of the Covid-19 dataset are formed around each Topic entity. There are 10 Topics: genomics, epidemiology, public-health-policies, lab-trials-human, healthcare-industry, clinical-treatment, virology, pulmonary-infections, influenza, and vaccines-immunology. By displaying the position of each of these topics to the predicted cluster embeddings, we can analyse the quality and quantity of their groupings.

The "find_cluster" module predicted clusters for the entities again but changed hyperparameters to K=10 and the initial distance of 30. Although ten groups are more than the optimal found with the Elbow Method, it is a good test to compare the task. The original plot had initial distances of 100 between the centroids, but this created too sparse of a graph, and with more embeddings, this would exacerbate the problem. Figure 27 displays the modified cluster embeddings plot.

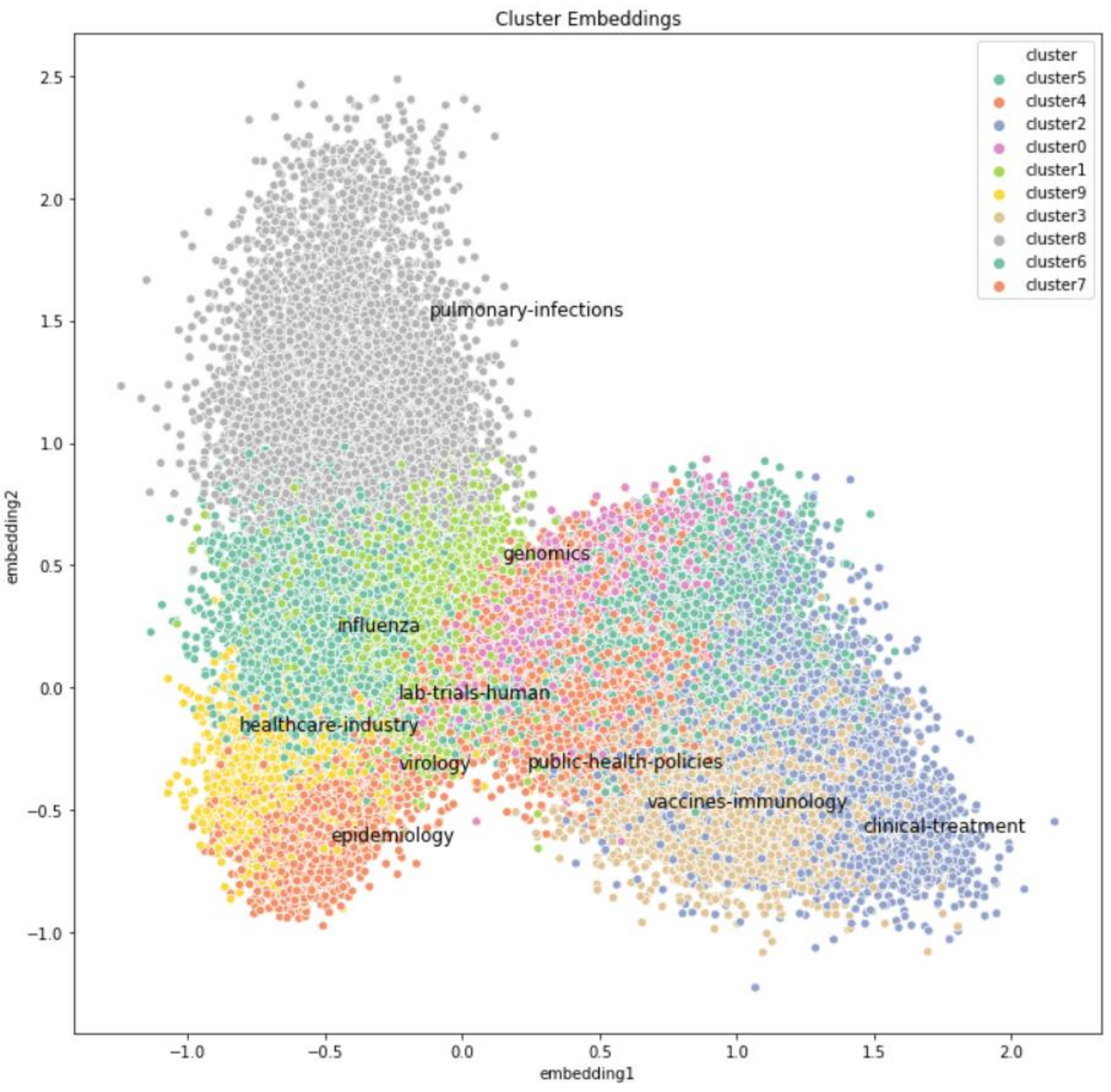
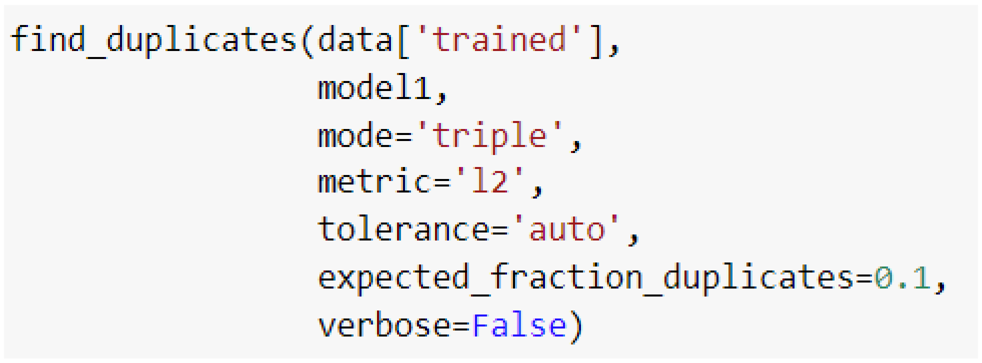


Figure 27 K-Means Clustering (K=10) compared to natural Topic groups

As seen the scatterplot, when there are ten different groups, some overlaps occur at the intersections. Some of the Topics are quite similar, which leads to crossovers, particularly between clusters 0, 1, 6, 7, and 9 in the middle. Influenza, virology, and epidemiology are related topics, and their respective predicted groups (0, 6, 7) display this. Likewise, lab-trials-human and healthcare-industry (groups 6 and 9) share many attributes. Some of the Topics are in the centre of their clusters with minimal overlap demonstrated by vaccines-immunology (3), clinical-treatment (2), and pulmonary-infections (8). The positioning of these embeddings, although not perfect, shows an accurate prediction representation of the natural topics and their associations. This clustering is an apt global assessment of the knowledge graph which shows the quality of the embeddings.

Module 3: Find Duplicates

Once the KGE model was made, the "find_duplicates" module can run. First, a test was completed with a subset of the dataset, shown in Figure 28.



```
find_duplicates(data['trained'],
                model1,
                mode='triple',
                metric='l2',
                tolerance='auto',
                expected_fraction_duplicates=0.1,
                verbose=False)
```

Figure 28 An example creation of the "find_duplicates" module

Here we search the triples in the subgroup for duplicates. The tolerance is the minimum distance between the embedded vectors that duplicate triples are accepted, in this instance set to automatic. This setting requires more computation as multiple tests are run for each triple with different tolerances to find the optimal. An optimisation algorithm uses a root-finding routine to compute the tolerance closest to the expected fraction (0.1). The 'metric' hyperparameter defines the distance algorithm, which calculates the similarity of elements. This example uses Euclidean distance (l2), but manhattan distance (l1) or minkowski_distance (l_p) can also calculate the spread between vectors. The output from running these settings was an automatic tolerance of 1.92 and multiple frozen sets of duplicate triples within this tolerance.

The mode was changed to 'entities' to get more meaningful output, and the tolerance was assigned to 1, using unique entities for the input data array. These new settings reduce the number of frozen sets returned as there are fewer elements to check with this mode. The runtime also decreases with a fixed tolerance that doesn't repeat tests, as automatic does. This test returned thousands of potential duplicates from the set of 140,000 unique entities. Every entity which had a least one similar entity within its threshold, calculated by the Euclidean distance between them, was added to a frozen set. The three different thresholds tested were 0.1, 0.5, and 1 to find which suited the dataset best. The 0.5 (219 duplicates) test was optimal as 0.1 had too few (2 duplicates found), and 1 had too many (over 1,000). Here is a description of two examples labelled as duplicates entities.

The two highest correlated (threshold <0.1) entity pairs found were:

1) {'1eb18d17-7fa9-4841-a5e0-d1a09242e5b3', '1ffa9e58-a466-48ff-9a55-efff6f580ea3'}

The first pair found in this range is of two Brazilian institutes called, Centro de Pesquisas Aggeu Magalhães in Recife, and CCD, Secretaria da Saúde in São Paulo, respectively. On this occasion, the two places aren't duplicates, as they are from two different regions in Brazil. They share one author who has an affiliation with both institutes, giving a short distance between their vector embeddings.

Although the filtration process in section 3.4 improved the dataset, during this testing, a flaw was found. Filtering removes entities which have less than five connections, but after their removal, other entities which linked to these have fewer connections. In this example, both institutes had four affiliated authors, and from each were removed. Both institutes have now only one relationship, which is shared, producing a very short distance between their embeddings even though they are not duplicates. This flaw gives a false positive output for the test, and therefore no extra relations should be added between these entities. These connections are plotted below in Figure 29.

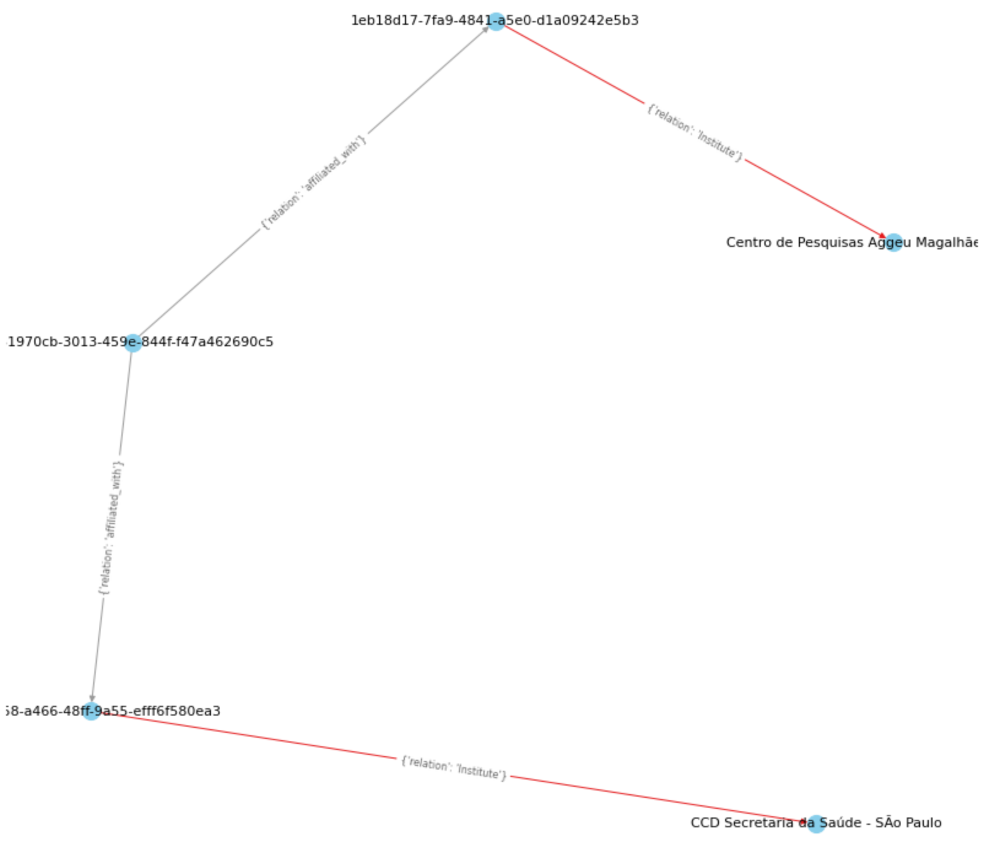

1eb18d17-7fa9-4841-a5e0-d1a09242e5b3
{'relation': 'Institute'}
{'relation': 'affiliated_with'}
Centro de Pesquisas Aggeu Magalhãe
1970cb-3013-459e-844f-f47a462690c5
{'relation': 'affiliated_with'}
58-a466-48ff-9a55-efff6f580ea3
{'relation': 'Institute'}
CCD Secretaria da Saúde - SÃo Paulo

Figure 29 "find_duplicates" example of institutes which are different

2) {'1c5b02a6-cac4-40d6-9299-02d405f82777', 'a58a72a4-2a78-4ae1-a959-d2b26b0fb6ef'}

These two unique IDs both link to authors, namely, C Tremblay and CÉCILE Tremblay, respectively. The two authors are the same person with a different spelling. Misspellings and variations in names can frequently occur in datasets, especially when data is extracted from the internet. Ampligraph's tool is an effective method of retrieving and fixing entities which are the same or very similar. Both author entities link to similar papers and institutes and therefore are in close Euclidean distance on the embedded vector space. This finding is a true positive assignment of duplicates which should be depicted in the knowledge graph. The best way to link them is by adding a new bidirectional relation "duplicateof" in the knowledge graph. A simple plot, created with NetworkX, displays these entities with their new connections in Figure 30. New links marked with red circles.

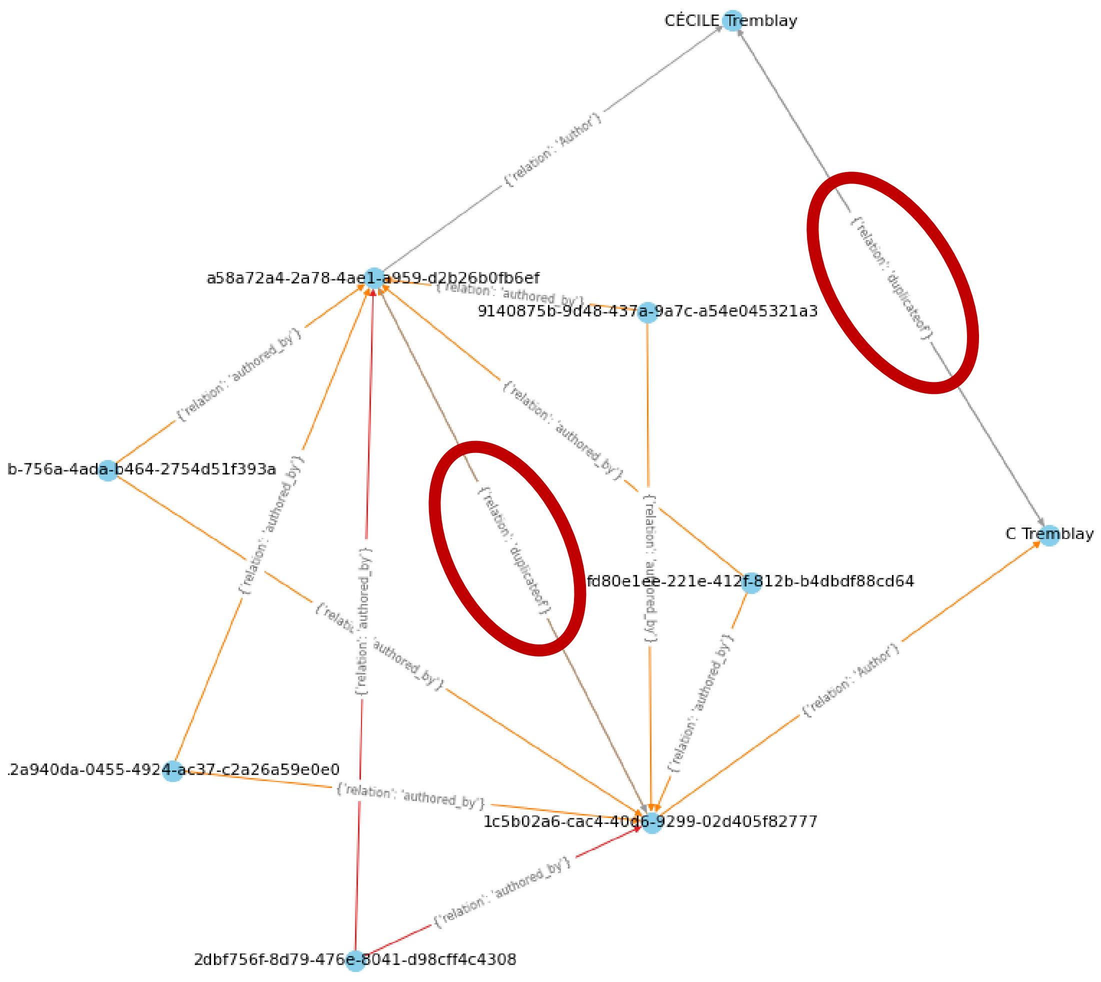

CÉCILE Tremblay
{'relation': 'Author'}
{'relation': 'duplicateof'}
a58a72a4-2a78-4ae1-a959-d2b26b0fb6ef
{'relation': 'authored_by'}
9140875b-9d48-437a-9a7c-a54e045321a3
b-756a-4ada-b464-2754d51f393a
C Tremblay
fd80e1ee-221e-412f-812b-b4dbdf88cd64
.2a940da-0455-4924-ac37-c2a26a59e0e0
1c5b02a6-cac4-40d6-9299-02d405f82777
2dbf756f-8d79-476e-8041-d98cff4c4308

Figure 30 "find_duplicates" example of authors which are the same

The benefit of finding these duplicates is to connect identical pairs in the knowledge graph. These are entities correlate positively to each other but were previously unknown. This module is a form of local KGC where the analysis is specific to regions in the graph. The result of adding links between these pairings gives extra information to the knowledge graph about related entities producing an interconnected the network that better represents the dataset.

5.3 Comparison

All of the approaches have their optimal use case and advantages. For KGC the "discover_facts" module works the most effectively at finding new links which can enhance the knowledge graphs performance. Over 1400 new triples were found which can aid in searching and discovery of related entities. The "find_clusters" method can accurately predict the groups of each entity to analyse the accuracy of the embeddings, with the help of Scikit-learn clustering libraries. This clustering is a crucial evaluation of the overall embedding and offers insight into the comprehension of the model in comparison with real-world classes. The final module tested "find_duplicates" is similar to the first method but retrieves entities which are potentially the same. It is essential to connect or remove duplicates as they add unnecessary complexity to the knowledge graph.

The visualisations provided with each created with different libraries enable an explainable comprehension of the connections in the knowledge graph. It is possible to understand the reason for predictions using these by focusing into particular regions. These plots not only allow awareness to expert users but can provide information to everybody, notably when represented in a visually pleasing way.

The overall impact of this chapter has a significant influence on the knowledge graph and users understanding of its mechanisms. KGC helps to improve the quality and completeness of a KG by finding relationship patterns which were not already known. Predictive accuracy and speed can vastly improve with the use of these methods, particularly with link prediction. These modules add Explainability to the dataset, which can help clarify decisions which KGE models make. This is most effective with the use of visual representation, which chapter 6 will investigate further.

6 Explainability

6.1 Overview

Chapter 6 is the final methodology section, which discusses Explainability. There are two other parts in this chapter to examine and implement explainable KGE. Section 6.2 discusses what explainability means, its importance, and approaches to use it with our KGE models. Section 6.3 follows on from this, to provide details of how we applied explainability to knowledge graphs, describing methods and assessments used.

6.2 Explainable knowledge graph embeddings

Explainability is a problem in AI which needs to be better understood and tackled. This problem is critical in the case of perilous tasks such as medical operations/diagnoses and driverless vehicles. These types of assignments are predominantly trained on Deep Learning algorithms which can produce unpredictable results and have an incomprehensible decision process, even to expert humans. Explainable AI (XAI) is not just a desirable characteristic, but one which is necessary for the future progression of AI.

XAI can be implemented in a variety of ways to give more interpretation to users about the decisions these algorithms make. There two main types of interpretability currently available. The first approach is a combination of black-box systems, with high accuracy, and a second model to retrieve information about the model's decisions, to allow for further analysis. The second approach is through the creation of a single transparent model which, through its design, is a natively explainable network. This project focuses on building inherently transparent models, but it would be possible to integrate our approaches with ML models in the future.

There are two types of analysis which can be useful depending on the task, Global and Local analysis. Guidotti et al. [68] discuss the dimensions of interpretability where a model can be inspected as a full system (Global), and through the individual decisions that the model makes (Local). Global explanation is much more challenging to achieve but can be analysed through clustering methods as seen in KGC. Local analysis interprets the decisions made by neighbouring entities, similar to the link prediction and duplication finding completed in section 5.2. Knowledge graph completion strongly relates to explainability as their shared goal is to find out more information about the dataset. Explanations can be given in a variety of formats including, a list of rules, visualisations, and comparisons.

Entity Resolution has seen some recent breakthroughs shown by Ebaid et al. [69] as they describe their new method EXPLAINER to understand ER classifiers and provide explanations for its decisions. They provide three levels of interpretation of models, namely, global, model-level, and classifier differences. Global is performed through cluster plots and feature importance. Model-level is implemented with visualisations and rule mining. Finally, classifier differences through comparisons of predictions and explanations. Similar to ER, knowledge graphs can perform this type of analysis on models through the use of KGE.

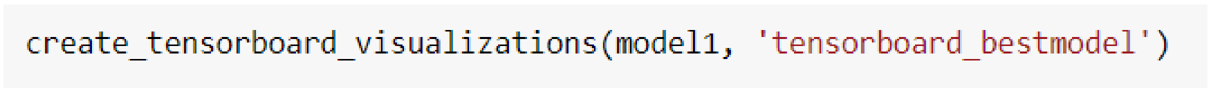
KGE are sub-symbolic as they no longer have the direct explanations which knowledge graphs contain, only embedded vector representations of them. The advantage of using this transparent approach, as opposed to an integrated system, is that the sub-symbolic KGE is not ambiguous because each embedding still relates to an element in the KG. Linking between the original KG and its embedding's predictions is key in interpreting its results. Bianchi et al. [70] released a paper this year, describing possible approaches for explainable KGE through logical reasoning and rule-mining. They state that explainability is low in current policies, and there is a need for future research.

The Covid-19 dataset, which the KGE models trained, is in crucial need of further interpretation. Chen et al. [71] discuss the importance and usefulness of knowledge graphs for coronavirus data, mentioning the CORD19 dataset explicitly, to understand the illness. They note how a KG from this dataset can answer research questions through visualisations and detailed interpretations. An example depicted in the paper examines the co-occurrence frequency of entities in the KG, showing how this knowledge is vital for medical researchers. The dataset is particularly useful as it is a semantic network of highly correlated information about a single topic. Providing explainability through KGE with this dataset helps to further this study.

Although chapter 5 details cluster and regional analysis of the KGE through KGC, precise information about the model's decisions still require further investigation. The next section 6.3 describes the process taken to achieve this.

6.3 Implementation

A further investigation of the embeddings can help to interpret more about the knowledge graph. One method of doing this is through three-dimensional visualisations which map the embeddings position. Tensorboard is a tool in Tensorflow which can do just this with its online platform ‘Embedding Projector’. The Embedding Projector allows the visualisation of high-dimensional data, with options available for different types of analysis of the embeddings. The embeddings must be exported from the Google Colab and uploaded to this platform. Ampligraph contains a utility function to help with this called “create_tensorboard_visualizations”. This function converts the embedded model in the file format, which is used by Tensorboard named Tab-separated values (TSV). Figure 31**Error! Reference source not found.** shows this function which converts “model1” into a list of different files needed to reproduce it in the “tensorboard_bestmodel” directory.



```
create_tensorboard_visualizations(model1, 'tensorboard_bestmodel')
```

Figure 31 Export embeddings to Tensorboard

“metadata.tsv” and “embeddings_projector.tsv” are the two main files created, both of which we uploaded to the Embedding Projector. The “embeddings_projector.tsv” file contains the 400 embedding vectors for each unique entity. The “metadata.tsv” stores the values which correspond to the name of each entity from the other TSV. Embedding Projector loads these files and maps them onto a three-dimensional space. There are three possible mapping methods which will now be described: Principal Component Analysis (PCA), UMAP, and T-SNE. Each way sphereizes the data by normalisation, shifting each entities centroid and making it the unit norm. This normalisation creates more comparable graphs between the methods.

Method 1: PCA

This PCA approach is similar to the clustering analysis in section 5.2.2, but in this instance, it is possible to investigate the further the relationships as there is an extra dimension added to the plot. The allows the envisage of connections between clusters which isn’t possible in two-dimensions. PCA concept is to re-align the axis in n-dimensional space to capture the highest amount of variance in the data. The three axes which represent the data are called the

Principal Components. PCA depicts the multicollinearity that changes features of the dataset into Principal Components which are unrelated but linearly linked to the data. The more variance about an axis, the more detail it holds. PCA reduces the dimensionality of while capturing the most information possible. PCA uses Eigen Decomposition, which is a matrix factorisation method, to find basis vectors which diagonalize the covariance matrix. The PCA implementation is approximate in Tensorboard 50,000 points in 200 dimensions are randomly sampled, but still represent the dataset effectively.

The defined three axes a percentage of variance, the combination of which is the variance of the graph. The X-axis has the highest variance, the Y-axis has the second-highest, and the Z-axis has the least and is dropped in two-dimensions. Figure 32 represents the PCA of the unique entities in the dataset, with a total of 92.1% variance. The percentages of variance per axis are X-axis 53%, Y-axis 35.6%, and Z-axis 3.5%.

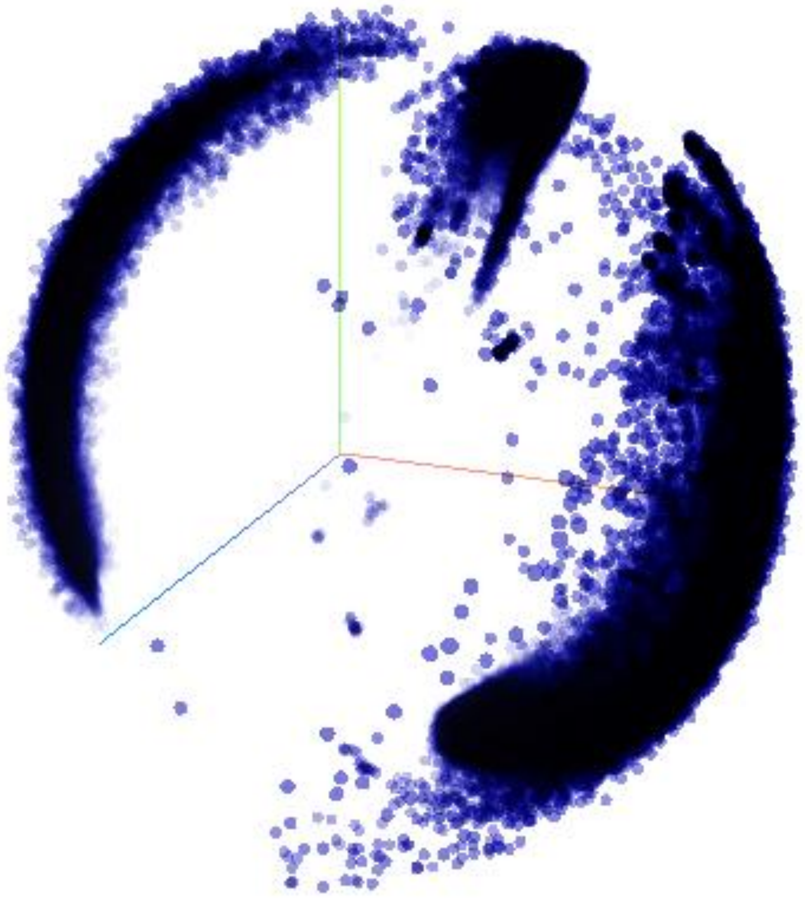
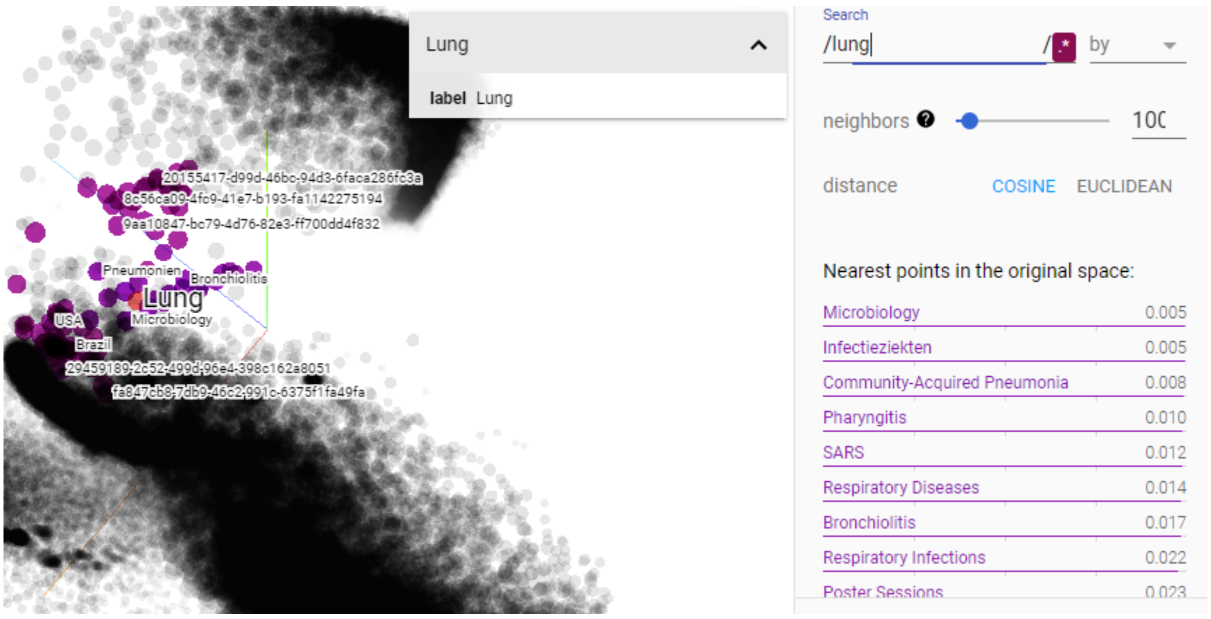


Figure 32 PCA 3-D model with 92.1% variance

This figure represents the clustered relationships present in the dataset. Each dot in the three-dimensional space relates to an entity. There are three large clusters around the axes which have subclusters present in each. The distinction between groups shows that the KGE model appropriately learned the connections between embedded vectors. Figure 32 is a form of global analysis of the dataset, but Tensorboard can also perform local interpretations.

Different segments of the plot can be captured to highlight similar embeddings. The graph can rotate, zoom, and emphasise any region of interest. As an example of this, the entity 'Lung' was searched for in the dataset. Our graph zoomed into its location, and the number of its nearest neighbours to highlight was set to 100. The cosine distance between vectors calculated the closest points. Figure 33 displays this example, showing the focused plot and a list of the nearest topics to 'Lung'. These relationships demonstrate the incredible accuracy to which the concepts are linked, particularly the most similar entities that match the real-world interpretation of related fields. The original KG does not contain this detailed information which adds Explainability to the predictions of the embedded models.


Lung
label Lung
20155417-d99d-46bc-94d3-6faca286fc3a
8c56ca09-4fc9-41e7-b193-fa1142275194
9aa10847-bc79-4d76-82e3-ff700dd4f832
Pneumonien
Bronchiolitis
Lung
Microbiology
USA
Brazil
29459189-2c52-499d-96e4-398c162a8051
fa847db8-7db9-46c2-991c-6375f1fa49fa
Search
by

/lung

neighbors 100

distance COSINE EUCLIDEAN

Nearest points in the original space:

Microbiology	0.005
Infectieziekten	0.005
Community-Acquired Pneumonia	0.008
Pharyngitis	0.010
SARS	0.012
Respiratory Diseases	0.014
Bronchiolitis	0.017
Respiratory Infections	0.022
Poster Sessions	0.023

Figure 33 PCA model with 'Lung' and nearest embedded vectors highlighted

Method 2: T-SNE

T-distributed Stochastic Neighbor Embedding (T-SNE) is an alternative approach to PCA for visualising data points training on ML algorithms. It is a nonlinear dimensionality reduction method which can map high-dimensional embedding data into lower dimensions. It models embeddings dependant on their vector space distance in two or three dimensions. T-SNE operates in two stages. The first makes a probability distribution over the dataset giving similar entities high probability measured with Euclidean distance. The second stage maps these high-dimensions over the distribution into a lower-dimensional space, minimising divergence between them.

Clusters can appear on the plot which don't represent the actual groupings as they strongly affected by the parameters set, perplexity, learning rate, and supervision. It is vital to understand what these are and how to define them correctly for the dataset. Otherwise, there is a risk of producing misleading relationships or misclassification of the data. Perplexity is the balance of focus between local and global features of the data. It depends on the density of the data, usually the larger the dataset, the more perplexity needed. Although if perplexity is too high clusters merge. The learning rate is also dependant on the data size; smaller datasets require a lower learning rate to train. The last metric is the T-SNE's supervision of the labelling while training scaled from 0(disabled)-100(complete importance).

The size of two clusters formed can look the same but have a considerable variation. The relative sizes of clusters are not visible with T-SNE as dense groups are expanded, and sparse groups are contracted, which displays them as similar sizes. Distances between separated clusters don't correlate to the magnitude of difference between topics globally. T-SNE is a flexible method of dimensionality reduction but can be hard to understand. With an understanding of this method and the dataset, it is possible to find out more information about it.

A range of perplexity values must be tested for T-SNE to correctly represents the data. Perplexity values of 2, 5, 30, 50, 100 were tested. Each ran for 2,000 iterations with a learning rate of 10. Figure 34 displays the results of these five tests.

From the resulting plots, we can see when perplexity is 2, the clusters are dense and separated widely, but there are many entities in the middle which haven't been clustered at all. These are signs that the T-SNE model has undertrained. Conversely, when the perplexity is too high at 100, some of the clusters appear much smaller in comparison to others meaning the model has overtrained. The plots when perplexity is 5 and 30 both are too sparsely spread and particularly at five the clusters are not well defined. The optimal perplexity value from this test is 50, which groups entities into distinct clusters without diminishing their sizes too much.

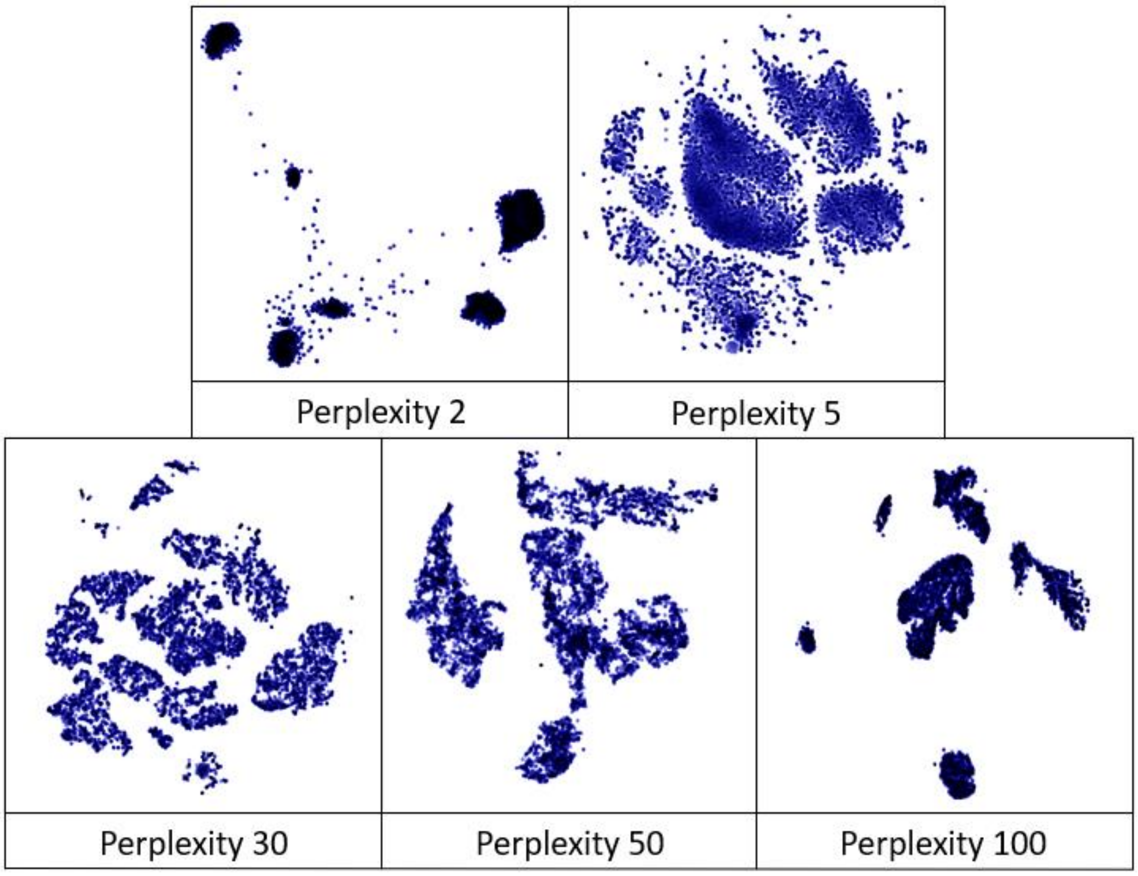


Figure 34 T-SNE test plots for different values of perplexity

T-SNE requires a lot of computation, and its global analysis leaves much to be desired concerning cluster sizes and distances. Multiple views at different perplexity values are necessary to find the optimal, but also to understand the topology of the model. It does, however, provide some intriguing visualisations which can flexibly explain the relationships from the embedded model. When focused into a particular node similar to PCA, it can infer local connections of neighbouring entities which it accurately maps. It can find structure from graphs which other dimensionality reduction algorithms can't. This characteristic makes it a technique which is always useful to test which can provide explainable information about knowledge graphs.

Method 3: UMAP

Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) is a third method for interpreting the embedded vectors. It is a fast and scalable mathematical method which uses manifold learning techniques and has many advantages over T-SNE.

T-SNE doesn't scale effectively for larger datasets as it requires a lot of computation. It also doesn't preserve the structure of the data globally, as stated in the previous method, cluster size and distance don't always represent the data. Essentially, it is hard to compare different clusters. T-SNE also must be reduced to lower dimensions with another algorithm to begin mapping, e.g. PCA. Finally, it is hard to tune T-SNE's hyperparameters and produce a visualisation which accurately represents the data.

UMAP can use any distance metric to calculate the difference in embedded vectors. UMAP doesn't apply normalisation to any probabilities, which dramatically reduces the runtime. Nearest neighbours parameter is used instead of the perplexity which T-SNE uses, which makes UMAP easier to tune. UMAP uses the BCE cost function instead of diversity which makes it better at demonstrating the global patterns in the data. T-SNE struggles to project large distances from high dimensions to low dimensions and preserves only the local structure correctly. UMAP also optimises better with SGD than T-SNE making it quicker to produce results.

The only parameter to set for UMAP is the number of neighbours. This parameter computes the fuzzy simplicial set to approximate the overall shape of the manifold. A simplicial set is a weighted graph, with edge weights representing the probability that two entities are linked. There are two stages of creating the UMAP. The first stage makes a topological representation. The second stage uses BCE to optimise the low dimensional model to represent the dataset correctly.

Three visualisations were created with UMAP with 15, 30, and 50 neighbours to represent the dataset. Figure 35 depicts this test.

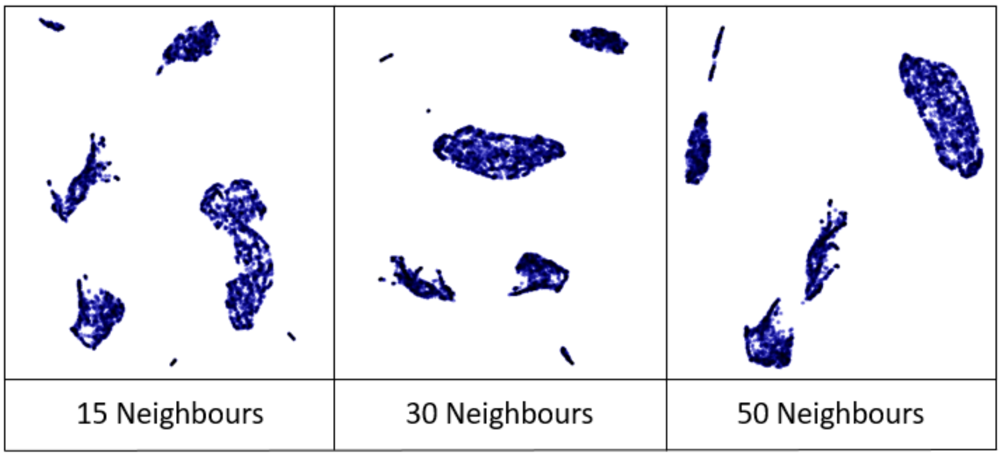


Figure 35 UMAP visualisation with different neighbour values

Choosing the neighbour radius is vital in creating an accurate plot with UMAP. If the radius is too small groups will be isolated, inversely too many neighbours will lead to poorly defined clusters. Each entity is linked to at minimum its closest neighbour with a 'fuzzy' feature which prevents the radius from increasing above a set threshold. The neighbour parameter balances the global and local design of the network. Figure 35 shows that neighbours of 15 and 30 are too low as their clusters are isolated with many outliers. Increasing the neighbour parameter above 50 will overlap groups, misrepresenting the KGE. When the neighbour radius is set to 50, there are five well-defined clusters with no clear outliers, making it the optimal tuning for the UMAP plot. These plots preserve the global structure of the knowledge graph embedding much more effectively than T-SNE. T-SNE struggles here mainly because its perplexity is too low to keep the system's shape and to increase it to higher levels requires too much computation.

Both T-SNE and UMAP distort the high dimensional embeddings which aren't directly interpretable like PCA. They do, however, provide an alternate view to PCA, which can help to explain the dataset better. All methods are powerful tools which help to visualise and interpret the relationships of high-dimensional data. These different interpretations give the user an advanced understanding of predictions made by the KGE model. Through visualisations, the explanations are accessible to all users but to understand the patterns of T-SNE and UMAP require expertise in this field. Although Explainability is not solved with these methods, they do provide more details about the datasets relationships, which was the aim of the project. Further analysis of explainable AI can be implemented with a combined integration between KGEs and DL models which could offer some ground-breaking results.

7 Results

7.1 Overview

Chapter 7 is designed to provide further interpretation of the results from each of the tasks, from chapters four to six. Section 7.2 discusses the KGE models detailing the reasons some models outperformed others. Section 7.3 further deciphers the KGC methods and their evaluation with the COVID-19 dataset. Finally, section 7.4 examines the explainability of the KGE gained from chapter 6.

7.2 KGE

Each model testing in chapter 4 produced results apart from ConvE, which was untrainable. The other three tested were TransE, DistMult, and ComplEx, and they all recorded adequate results when tested with evaluation metrics MR, MRR, and Hits@ rate. The original dataset was sparse and had too many unique entities, and none of the models achieved MRR scores above 0.05 when tested. This dataset was underperforming and is the reason the filtration process was added. When triples which contained entities with less than five connections were removed, then the number of unique entities reduced substantially. This process allowed the models to train more efficiently as fewer vectors were mapped, and the results of all the models improved.

With further hyperparameter tuning, an even more significant increase for the evaluation metrics was made, with the best models scoring over 0.25 on MRR and Hits@10. These scores plateaued at this point, and no further growth was possible. It is inconclusive as to why this happened, but the removal of triples from the knowledge graph with the filtration process might have been to blame. When unnecessary entities were removed so too were relations of connecting entities which reduced their semantic meaning. This procedure left many remaining entities with less than five links which an embedding model struggles to vectorise appropriately. A different type of filtration could have been more effective at reducing the knowledge graphs complexity and still keep its semantic information. Overall the KGE models trained were still powerful at prediction and could be used for many other applications.

7.3 Knowledge graph completion

Chapter 5 describes different methods of assessing and improving the KGE models. The first of these was to discover new facts about the knowledge graph with link prediction. This process achieved excellent results and found over 1400 new entities for the "associated_concept" and "similar" relation types. In addition to the findings, plots were created to understand why specific triples were ranked highly.

The second module tested clustered the data into groupings. With the help of the Elbow Methods analysis, it was deduced that nine clusters were optimal. As there are ten naturally occurring topics in the data, this method was clustered for ten different groupings for comparison. A visualisation of the embedded vectors with the positions of the topic entities was made to allow investigation. Almost all of the topics were separated into different clusters and were near the centre of their respective sets. Only a few cases overlapped the groups, but these were usually similar topics. This test offers a global analysis of the embeddings, which is useful for Explainability.

The final module tested found over 200 duplicates in the graph, which were deemed as copies. Some of these were correctly classified and were linked in the KG as duplicates. There were many others which were close together as they shared some connections and had others removed with filtration. These instances were misclassified and were ignored as duplicates. This process is necessary but slow to make sure the knowledge graph correctly represents the actual data.

7.4 Explainability

Chapter 6 offers some final insights into the predictions of the KGE and how individual relationships are depicted in the KG. Visualisations were plotted with Tensorboard to give more interpretation of embeddings, and three different dimensionality reduction algorithms were applied. These algorithms were PCA, UMAP, and T-SNE, which all attempted to globally and locally analyse the network. They all give a three-dimensional view of the embeddings, which wasn't possible with some of the clustering plots created in previous chapters. These 3-D graphs provide an alternative perspective about the embeddings and can help to explain predictions. The next level of Explainability, which future research could examine, is a detailed text-based description of predictions integrated with DL algorithms.

8 Conclusion and Future Works

8.1 Overall Contribution

Here is a comparison of the overall contributions made in this work to the research aims stated in Section 1.3:

1) Creating a knowledge graph, in the form of triples, linking between data nodes in a knowledge base (Section 3)

The first aim of this project was to create a knowledge graph from a large-scale semantic dataset. As demonstrated in Section 3, this project used an extracted Covid-19 dataset which compared the correlation between research papers, their authors and institutes, and related topics and concepts. With the assistance of AWS's CKG, this dataset was converted into a knowledge graph in the form of triples. With 11 different relation types, over 780,000 unique entities, and 3 million triples, this knowledge graph incorporated a massive network of linked nodes. When testing the KGE, this KG was too sparse, and a filtration process was added to reduce the number of unique entities.

The outcome of this filtration reduced the number of triples to 2.3 million and the number of unique entities to 140,000, making the knowledge graph denser and less variable. Less variety in the KG is a disadvantage, but it increases the prediction quality of KGE models which critical for KGs applications.

2) Apply knowledge graph embedding models to obtain vector representations of entities and relations (Section 4)

The second objective of this project was to create a knowledge graph embedding model to represent the KG produced in section 3. Chapter 4 discusses the types of KGE models and how to make them with the aid of the Python library Ampligraph, which creates vector representations of concepts in an embedded space. This library also contains evaluation methods to test the accuracy of embedded models.

Four different KGE models were trained under several hyperparameter settings aiming to tune each optimally. A comparison of their evaluation results showed that each performed to a satisfactory standard, but the ComplEx-N3 model was most effective for the dataset. Mean Rank, Mean Reciprocal Rank, and Hits@n ranking were the metrics implemented to compare the models. The ComplEx-N3 model boasts an MR of 2104, MRR of 0.25, and Hits@10 rank of 0.27. These metrics prove that the embedded model accurately learns the knowledge graph's network of links.

3) Link prediction methods are performed for KG completion, choosing the best performing - highest accuracy and broadest variety (Section 5)

The third aim of the project was to perform KGC using link prediction on the best performing KGE model. The ComplEx-N3 model, which performed best in chapter 4, was used to predict new links in the KG. Ampligraph's Discovery module assisted in the retrieval of newly predicted links as it has a method for discovering facts about a knowledge graph. The highest-ranking new triples were added to the KG for completion to improve the accuracy and variety of connections. Over 1,400 new triples for one of the methods which add more information to the KG. The clustering analysis compares the natural topic entities with the predicted clusters in the graph. This method allows an analysis of how well the embeddings represent the data from the KG. The third method found over 200 duplicates, and the positive connections were added to the KG to describe these.

4) Using this knowledge graph embedded model to create predictions and interpretability of these predictions (Section 6)

The final objective of this project was to provide Explainability for the knowledge graph. With the assistance of some KGC tools, interpretability about the network was found through visualisations, the discovery of new relationships, and the prediction of duplicate entities. The Tensorboard provided a novel perspective on the relationships between embeddings. A greater understanding of the decisions the model makes can be determined using these visualisations. They provide a crucial insight into interpreting the dataset, which wasn't possible before. Although this form of Explainability is only applicable to expert users, the same techniques could help to enhance this further.

8.2 Conclusion

Explainability for artificial intelligence is a challenging but vital problem to solve as it is necessary to gain the trust of humans. XAI is essential when tasks are particularly hazardous to people's lives. As new technology develops, academic research must understand how these autonomous devices make their decisions. Unfortunately, the most effective AI algorithms are opaque black-box systems which even expert designers don't understand. The lack of transparency can induce biases, especially for socially sensitive tasks. A justification of behaviour helps prevent these unfair biases of a system's decisions.

Through the study of the topic, I learned that many modern applications are becoming increasingly difficult for comprehension. This project aimed to use transparent models to develop explanations about predictions. Through interpretation and visualisations, Explainability was successfully added to KGE while keeping accuracy consistent. This type of research will have a significant social impact in the future as AI algorithms become more prevalent in risky applications.

From a Covid-19 perspective, this investigation had the ambition to provide valuable information about the virus from research papers. Through the use of knowledge graphs and their embeddings, it was possible to create a network of interconnection relationships which represent the data. As seen in the clustering analysis, it effectively maps out a distributed view of the data. These models can have many applications for retrieving information, relationship matching, and user comprehension of the virus.

8.3 Limitations

With every project, there are limitations, and this one is no exception. A wide variety of processes could have been implemented differently throughout the project. Some methods were not implemented, which could have added value to this research. The primary reasons for these limitations are constraints, including the project's time-scale, the available equipment, and restraints of approaches themselves. Despite the projects research contributions, this section details some of its disadvantages.

The first limitation, although minor, was from the filtering phase of the dataset, section 3.4. This phase was needed to reduce sparsity in the embedded model, but it came at a cost. By implementing this over 600,000 unique entities and 1 million triples were removed from the knowledge graph. This removal of nodes reduced the number of connected entities creating complications with similarity in the vector space seen in section 5.2.2 when false duplicates are predicted. The variety and range of data which the KG represented were reduced dramatically with this change, but with minimal links between individual entities, the embedded model can’t learn effectively. The trade-off between accuracy and information was necessary to make here, but further filtering could have skewed results.

ConvE was one of the four KGE models tested in chapter 4. It was the only model which didn’t produce any results because we were unable to train it with the dataset. The exact reason for why the model didn’t work is unknown, but we deduced that it was due to the size of unique entities in the knowledge graph which put too much pressure on the GPU available. Even when a subset of the data was used, it crashed Google Colabs 15GB GPU. Taking a tiny batch of the dataset did train, but it couldn’t be compared with the other models as it didn’t represent the dataset effectively. If more resources were available, such as a larger GPU, then potentially it would have trained, but this wasn’t the case.

Unfortunately, no Graph feature KGE models were tested. As determined in the literature review, neither latent of graph feature models were superior, and both could be useful depending on the task. There were no Graph feature models available in Ampligraph’s library, and implementing other libraries would add unnecessary work this projects section. Some research was put into testing Node2Vec model [72], but because of time restraints and the lack of comparable functions, no model was used in testing.

Training the KGE models was a difficult task which was dependant on the hyperparameters and loss function chosen. These settings have a significant influence on the prediction accuracy in testing, which made their tuning to optimal parameters tedious and laborious. This issue made it hard to improve accuracy as many models were trained with different hyperparameters only to see slight increases. This limitation is due to the KGE model’s evaluation process, which should be revised.

Overall the models underperformed in comparison to state-of-the-art results obtained from the benchmark datasets reviewed. No model substantially outperformed the others, all receiving MRR and Hits@10 scores over 0.2, but only ComplEx-N3 scored higher than 0.25 on either metric. These unnoteworthy results were primarily the fault of the vast number of unique entities in the graph, making predictions more challenging.

Link Prediction for Knowledge graph embeddings is uncalibrated [70]. Essentially, there is no way to compare the results of link prediction across models or even to determine how reliable they are. This limitation affects the results when specifying a relation to find new links when there are several different types. A calibration of link prediction is necessary for the trustworthiness of the output. A potential solution is to add some scaling to models to allow comparisons of this task.

The Explainability model wasn't implemented into an integrated approach with other ML algorithms. The KGE models were used to test link prediction and interpretability for the dataset, which is an inherently transparent model. This method does sacrifice the accuracy of performance for clarity in the model. KGE embedded models are useful for semantic datasets, but for image classification tasks would struggle to score adequately. A combination of Deep Learning and KGE models would considerably improve the quality of predictions and representation of the data. Regrettably, there wasn't enough time to create this type of integrated system, but it should be examined for future works.

The Explainability provided in chapter 6 is only comprehendible to expert users. A fully developed platform is needed to give more explanation, which could describe why specific decisions were made. This explanation would be written in layman's terms, with less technical detail, so that it could be interpretable to all users. The overall level of explainability is still low, and the models need enhancements to give the level of interpretation which users require to trust AI models.

8.4 Future Works

In this thesis, knowledge graph embedded models have been implemented for link prediction and explainability tasks. We discussed in detail the KGE approach to provide explainable AI through the use of visualisations and interpretations of entities in the knowledge graph. This low dimensional view of the knowledge graph has provided a more in-depth analysis of the dataset. In particular, the use of the Covid-19 research data can provide valuable information about the nature of the disease by efficiently finding relationships and interpretations about published papers. This research may help to battle the illness through the perspective of increasing people's knowledge with accurate material.

Further work could continue to use KGEs implementing explainable AI by providing full integration between knowledge graphs and machine learning models. An integrated model could improve the explainability of opaque black-box deep learning algorithms by providing a semantic analysis of the relationships in a dataset. Other possible research could test the quality of other KGEs, including Graph feature models, on these tasks to see discrepancies of results. The higher the ranking accuracy of a model, the better it will perform on other applications.

Future research of other approaches such as Entity Resolution and Probabilistic Soft Logic could attempt to implement explainability for ML models. Comparisons of their results with those of KGE can show which is most useful for a particular task. An investigation into the similarities of these methods could help to create combined solutions.

Research of COVID 19 is currently of considerable significance to society. Therefore, data processing investigation is invaluable to the medical and science communities, for their perception of the disease. Following this project, and the thousands of others which are being explored, the world hopes to subdue this virus. To accomplish this, a broad spectrum of fields must investigate their latest technologies. The cooperation and collaboration shown between different areas, especially information technology and medical science, should be emulated in the future to conquer other problems.

Explainability is a difficult problem to solve in artificial intelligence, but it is necessary for its next stage of development. Understanding how and why AI algorithms make individual decisions is crucial in gaining our trust. As more complex algorithms evolve, so too will the need for more comprehensive, explainable systems to interpret them. When most people imagine the future of AI, prediction accuracy is the prevailing notion of exemplary standards, but new measures need to incorporate the importance of transparency of these networks.

References

[1] N. Bostrom, Superintelligence: Paths, Dangers, Strategies, New York: Oxford University Press, 2014.

[2] A. Singhal, “Introducing the Knowledge Graph: Things, Not Strings,” [Online]. Available: https://www.blog.google/products/search/introducing-knowledge-graph-things-not/. [Accessed 22 05 2020].

[3] D. Gunning, “Explainable artificial intelligence (xai),” *Defense Advanced Research Projects Agency (DARPA), nd Web,* vol. 2, 2017.

[4] N. Guan, D. Song and L. Liao, “Knowledge graph embedding with concepts,” in *Knowledge-Based Systems*, Elsevier, 2019, pp. 38-44.

[5] W. Quan, Z. Mao, B. Wang and L. Guo, “Knowledge graph embedding: A survey of approaches and applications,” *IEEE Transactions on Knowledge and Data Engineering,* vol. 29, no. 12, pp. 2724-2743, 2017.

[6] M. Nickel, K. Murphy, V. Tresp and E. Gabrilovich, “A Review of Relational Machine Learning for Knowledge Graphs,” *Proceedings of the IEEE,* vol. 104, no. 1, pp. 11-33, 2016.

[7] K. S. a. W. Wang, “Building Trust in Artificial Intelligence,Machine Learning, and Robotics,” *CUTTER BUSINESS TECHNOLOGY JOURNAL,* pp. 47-53, 2018.

[8] S. S. T. R. B. Derek Doran, “What Does Explainable AI Really Mean? A New Conceptualization of Perspectives,” 2 October 2017. [Online]. Available: https://arxiv.org/abs/1710.00794. [Accessed 21 05 2020].

[9] J. F. Allen and A. M. Frisch, “What's in a Semantic Network?,” in *Proceedings of the 20th Annual ACL Meeting, Assoc. for Computational Linguistics*, New York, 1982.

[10] S. Nurdiati and C. Hoede, “25 YEARS DEVELOPMENT OF KNOWLEDGE GRAPH THEORY: THE RESULTS AND THE CHALLENGE,” 2008.

[11] L. Ehrlinger and W. Wöß, “Towards a Definition of Knowledge Graphs,” in *In Proceedings of the SEMANTICS 2016*, Leipzig, Germany, 2016.

[12] M. Richardson and P. Domingos, “Markov Logic Netowrks,” *Machine Learning,* vol. 62, p. 107–136, 2006.

[13] L. Mihalkova and R. J. Mooney, “Bottom-Up Learning of Markov Logic Network Structure,” in *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[14] M. Jaeger, “Relational Bayesian Networks,” *ArXiv,* vol. abs/1302.1550, pp. 266-273, 1997.

[15] I. Sutskever, R. Salakhutdinov and J. Tenenbaum, “Modelling Relational Data using Bayesian Clustered,” in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[16] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston and O. Yakhnenko, “Translating embeddings for modeling multi-relational data,” in *Advances in neural information processing systems*, Advances in neural information processing systems., 2013, pp. 2787-2795.

[17] L. Lüa and T. Zhou, “Link prediction in complex networks: A survey,” *Physica A: Statistical Mechanics and its Applications,* vol. 390, no. 6, pp. 1150-1170, 2011.

[18] L. Galárraga, C. Teflioudi, K. Hose and F. M. Suchanek, “AMIE: association rule mining under incomplete evidence in ontological knowledge bases.,” in *Proceedings of the 22nd international conference on World Wide Web*, 2013.

[19] N. Lao and W. W. Cohen, “Relational retrieval using a combination of path-constrained random walks,” *Machine Learning,* vol. 81, no. 1, pp. 53-67, 2010.

[20] M. Nickel, V. Tresp and H.-P. Kriegel, “A Three-Way Model for Collective Learning on Multi-Relational Data,” in *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[21] T. Doghri, L. Szczecinski, J. Benesty and A. Mitiche, “Bilinear Models for Machine Learning,” in *Institut National de la Recherche Scientifique*, 2019.

[22] D. Krompaß, M. Nickel, X. Jiang and V. Tresp, “Non-Negative Tensor Factorization with RESCAL,” in *ECML/PKDD 2013 Workshop on Tensor Methods for Machine Learning*, 2013.

[23] S. Riedel, L. Yao, A. McCallum and B. M. Marlin, “Relation Extraction with Matrix Factorization and Universal Schemas,” in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.*, 2013.

[24] F. M. Suchanek, G. Kasneci and G. Weikum, “Yago: A Core of Semantic Knowledge,” in *16th International Conference on the World Wide Web*, 2007.

[25] N. Maximilian, V. Tresp and H.-P. Kriegel, “Factorizing yago: scalable machine learning for linked data,” in *Proceedings of the 21st international conference on World Wide Web.*, 2012.

[26] R. Jenatton, N. L. Roux, A. Bordes and G. R. Obozinski, “A latent factor model for highly multi-relational data,” *Advances in neural information processing systems,* pp. 3167-3175, 2012.

[27] D. Lucas, S. Rendle and L. Schmidt-Thieme, “Predicting RDF triples in incomplete knowledge bases with tensor factorization,” in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, 2012.

[28] S. Rendle and L. Schmidt-Thieme, “Pairwise interaction tensor factorization for personalized tag recommendation,” in *Proceedings of the third ACM international conference on Web search and data mining*, 2010, pp. 81-90.

[29] S. Rendle, “Scaling factorization machines to relational data,” *Proceedings of the VLDB Endowment,* vol. 6, no. 5, pp. 337-348, 2013.

[30] P. Miettinen, “Boolean tensor factorizations,” in *2011 IEEE 11th International Conference on Data Mining*, 2011.

[31] Q. Wang, Z. Mao, B. Wang and L. Guo, “Knowledge graph embedding: A survey of approaches and applications,” *IEEE Transactions on Knowledge and Data Engineering,* vol. 29, no. 12, pp. 2724-2743, 2017.

[32] L. Chang, M. Zhu, T. Gu, C. Bin, J. Qian and J. Zhang, “Knowledge graph embedding by dynamic translation,” *IEEE Access,* vol. 5, pp. 20898-20907, 2017.

[33] M. Fan, Q. Zhou, E. Chang and T. F. Zheng, “Transition-based knowledge graph embedding with relational mapping properties,” in *Proceedings of the 28th Pacific Asia Conference on Language, Information and Computing*, 2014.

[34] Z. Wang, J. Zhang, J. Feng and Z. Chen, “Knowledge graph embedding by translating on hyperplanes,” in *Twenty-Eighth AAAI conference on artificial intelligence*, 2014.

[35] Y. Lin, Z. Liu, M. Sun, Y. Liu and X. Zhu, “Learning entity and relation embeddings for knowledge graph completion,” in *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[36] B. Yang, W.-t. Yih, X. He, J. Gao and L. Deng, “Embedding entities and relations for learning and inference in knowledge bases,” *arXiv,* 2014.

[37] L. Galárraga, C. Teflioudi, K. Hose and F. M. Suchanek, “Fast rule mining in ontological knowledge bases with AMIE+,” *The VLDB Journal,* vol. 24, no. 6, pp. 707-730, 2015.

[38] N. Lao, T. Mitchell and W. W. Cohen, “Random Walk Inference and Learning in A Large Scale Knowledge Base,” in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, Edinburgh, 2011.

[39] M. Nickel, X. Jiang and V. Tresp, “Reducing the rank in relational factorization models by including observable patterns,” in *Advances in Neural Information Processing Systems*, 2014, pp. 1179-1187.

[40] A. Joulin, E. Grave, P. Bojanowski, M. Nickel and T. Mikolov., “Fast Linear Model for Knowledge Graph Embeddings,” *arXiv preprint arXiv:1710.10881,* 2017.

[41] M. Nickel, L. Rosasco and T. Poggio, “Holographic embeddings of knowledge graphs,” in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[42] T. Trouillon, J. Welbl, S. Riedel, E. Gaussier and G. Bouchard, “Complex embeddings for simple link prediction,” in *International Conference on Machine Learning (ICML)*, 2016.

[43] T. Trouillon and M. Nickel, “Complex and holographic embeddings of knowledge graphs: a comparison,” *arXiv preprint arXiv:1707.01475,* 2017.

[44] T. Dettmers, P. Minervini, P. Stenetorp and S. Riedel, “Convolutional 2d knowledge graph embeddings,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[45] D. Q. Nguyen, T. D. Nguyen, D. Q. Nguyen and D. Phung, “A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network,” in *arXiv*, 2017.

[46] C. Fellbaum, WordNet: An Electronic Lexical Database., Cambridge, MA: MIT Press., 1998.

[47] H. Bast, F. Bäurle, B. Buchhold and E. Haußmann, “Easy access to the freebase dataset,” in *Proceedings of the 23rd International Conference on World Wide Web*, 2014.

[48] D. L. GETOOR and D. A. MACHANAVAJJHALA, “Entity Resolution for Big Data,” Data Community DC, 13 8 2013. [Online]. Available: http://www.datacommunitydc.org/blog/2013/08/entity-resolution-for-big-data. [Accessed 9 8 2020].

[49] A. K. Elmagarmid, P. G. Ipeirotis and V. S. Verykios, “Duplicate record detection: A survey,” *IEEE Transactions on knowledge and data engineering,* vol. 19, no. 1, pp. 1-16, 2006.

[50] H. Köpcke, A. Thor and E. Rahm, “Evaluation of entity resolution approaches on real-world match problems,” *Proceedings of the VLDB Endowment,* vol. 3, no. 1-2, pp. 484-493, 2010.

[51] M. Brocheler, L. Mihalkova and L. Getoor, “Probabilistic similarity logic,” *arXiv preprint arXiv:1203.3469,* 2012.

[52] T. Trouillon, C. R. Dance, E. Gaussier, J. Welbl, S. Riedel and G. Bouchard, “Knowledge graph completion via complex tensor factorization,” *The Journal of Machine Learning Research,* vol. 18, no. 1, pp. 4735-4772, 2017.

[53] K. Hayashi and M. Shimbo, “On the Equivalence of Holographic and Complex embeddings for link prediction,” *arXiv preprint arXiv:1702.05563,* 2017.

[54] H. Xiao, M. Huang and X. Zhu, “From one point to a manifold: Knowledge graph embedding for precise link prediction,” *arXiv preprint arXiv:1512.04792,* 2015.

[55] B. Shi and T. Weninger, "Open-World Knowledge Graph Completion," *ArXiv,* 2018.

[56] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak and Z. Ives, "DBpedia: a nucleus for a web of open data," in *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, 2007.

[57] R. Catherine, K. Mazaitis, M. Eskenazi and W. Cohen, "Explainable entity-based recommendations with knowledge graphs," *arXiv preprint arXiv:1707.05254,* 2017.

[58] X. Wang, D. Wang, C. Xu, X. He, Y. Cao and T.-S. Chua, "Explainable reasoning over knowledge graphs for recommendation.," in *Proceedings of the AAAI Conference on Artificial Intelligence pp5329-5336*, 2019.

[59] R. Goebel, A. Chander, K. Holzinger, F. Lecue, Z. Akata, S. Stumpf, P. Kieseberg and A. Holzinger, "Explainable AI: the new 42?," in *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, 2018.

[60] L. L. Wang, K. Lo, Y. Chandrasekhar, R. Reas, J. Yang, D. Eide, K. Funk, R. Kinney, Z. Liu and W. Merrill, "CORD-19: The COVID-19 Open Research Dataset," *ArXiv,* 2020.

[61] Allen Institute For AI, "COVID-19 Open Research Dataset Challenge (CORD-19)," 2020. [Online]. Available: https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge. [Accessed 7 22 2020].

[62] AWS, "Building and querying the AWS COVID-19 knowledge graph," Amazon Web Services, 2020. [Online]. Available: https://aws.amazon.com/blogs/database/building-and-querying-the-aws-covid-19-knowledge-graph/. [Accessed 22 07 2020].

[63] L. Costabello, S. Pai, C. L. Van, R. McGrath, N. McCarthy and P. Tabacof, "AmpliGraph: a Library for Representation Learning on Knowledge Graphs," Accenture Labs Dublin, March 2019. [Online]. Available: https://docs.ampligraph.org/en/1.3.1/. [Accessed 29 07 2020].

[64] Google Brain Team, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," Google, [Online]. Available: https://www.tensorflow.org/. [Accessed 29 07 2020].

[65] L. Costabello, S. Pai, C. L. Van, R. McGrath, N. McCarthy and P. Tabacof, "Ampligraph Performance," Accenture Labs Dublin, [Online]. Available: https://docs.ampligraph.org/en/latest/experiments.html. [Accessed 29 07 2020].

[66] T. Lacroix, N. Usunier and G. Obozinski, "Canonical tensor decomposition for knowledge base completion," *arXiv preprint arXiv:1806.07297,* 2018.

[67] scikit-learn, "Scikit-learn: Machine Learning in Python; 2.3. Clustering," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[68] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, D. Pedreschi and F. Giannotti, “A survey of methods for explaining black-box models,” in *ACM computing surveys (CSUR)*, New York, 2018.

[69] A. Ebaid, S. Thirumuruganathan, W. G. Aref, A. Elmagarmid and M. Ouzzani, “Explainer: Entity resolution explanations,” in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019.

[70] F. BIANCHI, G. ROSSIELLO, L. COSTABELLO, M. PALMONARI and P. MINERVINI, “Knowledge Graph Embeddings and Explainable AI,” in *arXiv preprint arXiv:2004.14843*, 2020.

[71] C. Chen, I. A. Ebeid, Y. Bu, and Y. Ding, “Coronavirus Knowledge Graph: A Case Study,” *arXiv preprint arXiv:2007.10287,* 2020.

[72] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016.

[73] Q. Wang, J. Wiu, Y. Luo, B. Wang and C.-Y. Lin, “Knowledge Base Completion via Coupled Path Ranking” [Online]. Available: https://slideplayer.com/slide/13083351/. [Accessed 31 07 2020].