UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

THE IMPACT OF GRAPH EMBEDDINGS ON ENHANCING MACHINE LEARNING MODELS

LIAM JAMES GLENNIE ENGLAND

Thesis supervisor
ANNA QUERALT CALAFAT (Department of Service and Information System Engineering)

Degree
Master's Degree in Data Science

Master's thesis

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

26/06/2024

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Anna Queralt Calafat, for her constant support and invaluable feedback throughout this thesis. Her guidance was essential to the success of this work.

I am also thankful to my friends and family for their unwavering support and for providing much-needed distractions when necessary. Their encouragement and patience were vital in helping me complete this thesis.

Abstract

This thesis investigates the enhancement of poor-quality datasets, characterised by a limited number of features or features poorly related to a machine learning problem, by integrating knowledge graph embeddings to boost the performance of machine learning models in classification and regression tasks. The study introduces an extension of OWL2Vec* and assesses the effectiveness of the TransE and extended OWL2Vec* embeddings, revealing that their usefulness for regression and classifications tasks is highly dependent on the presence and representation of correlated features within the knowledge graph. Findings indicate that OWL2Vec* is more versatile when interpreting information in different forms, being most effective in handling relevant features as classes or attributes. On the other hand, results show that TransE understands relationships between entities better than OWL2Vec*. The research underscores the significance of meticulously designed knowledge graphs and proposes future work to achieve potentially superior embeddings.

1 Table of Contents

Contents

List of Figures

List of Tables

2 Introduction

In recent years, there has been a surge in knowledge graph usage leading to extensive research into their applications, such as search engines, recommender systems, question-answer systems, drug discovery, and fraud detection. Knowledge graphs (KGs) are directed labelled graphs that can be formed from information from a single source or combine information from multiple sources to represent some knowledge. As operations on large-scale KGs can be expensive, there has been a influx in knowledge graph embedding algorithms. Graph embeddings are vector representations of the graph capable of summarising the information found in the KG, which can be used for tasks like link prediction, knowledge graph completion, and entity resolution.

The primary aim of this thesis is to enhance poor-quality datasets, characterised by a limited number of features or features poorly related to the machine learning problem, by enriching them with knowledge graph embeddings related to the datasets. Machine learning models are known to struggle on datasets with a lack of features or irrelevant features. Therefore, the graph embeddings are expected to improve predictive performance by incorporating semantics, relationships between entities of the dataset, and relevant features from the KG entities. This additional information provides the model with context that might otherwise be lacking. The graph embeddings will be integrated with the original dataset features to predict labels within the dataset.

Thus, the main research objectives are:

- **RO1:** To evaluate whether graph embeddings can enhance the performance of machine learning models in classification and regression tasks when dealing with poor-quality datasets.
- **RO2:** To assess whether the embedding algorithms can be improved for classification and regression tasks.
- **RO3:** To analyse how graph embedding algorithms interpret relevant data within the graph and to investigate how the quality of the embeddings are impacted by the form in which the data is represented in the knowledge graph.

To the best of current knowledge, there is no existing literature that combines graph embeddings with tabular data to enhance predictive performance in machine learning tasks like classification and regression tasks. This gap underscores the importance of this research, which aims to expand the scope of knowledge graphs and their embeddings.

This thesis uses datasets containing URIs (Universal Resource Identifier) for every instances. Thus, each instance can be linked to a node within a knowledge graph because they share the same identifier, URI. The datasets are combined with graph embeddings, and extensive experiments have been conducted to understand the functionality of these algorithms and their data interpretation. Some of the experiments include modifications to the training graphs and extensions to graph embedding algorithms to potentially enhance embedding quality.

The thesis is structured as follows: the Motivation section elaborates on how graph embeddings can improve classification and regression models for poor-quality datasets, providing a hypothetical example in which embeddings could exploit the semantic information and relationships of the graph. The Related Work section introduces relevant literature and identifies a gap that this thesis aims to fill. The Background section offers an overview of key concepts, including knowledge graphs, description frameworks, and embedding algorithms used. The Methodology section details the datasets, the constructions of knowledge graphs and their modifications, embedding training methods, our extension of embedding algorithms, and the training and evaluation of classification and regression tasks. Finally, the thesis presents the results, followed by

conclusions and future work. The code and graphs used in this thesis can be found in GitHub

3 Motivation

Graph embeddings have emerged as a potent tool in machine learning, especially for tasks involving graph-structured data. They encode graph, structural and relational information into low-dimensional vectors, facilitating tasks like link prediction, node classification, and community detection. However, while much focus has been on evaluating their performance in these tasks, a significant gap remains in understanding their potential in improving classic machine learning tasks such as classification and regression of external data related to the graph, like predicting the fuel efficiency of car models or the market value of top Forbes companies.

Traditional machine learning often struggles with sparse or poor-quality datasets, where features may be scarce, lack clear patterns, or exhibit weak relationships. In such scenarios, graph embeddings offer promise by enriching data representation through capturing entity relationships, structural, and semantic information inherent in the knowledge graph. Graph embeddings could also include attributes that are related to the target to be predicted, such as, company profits or number of employees. By embedding nodes and edges into continuous vector spaces, they enable the integration of graph-based information with traditional machine learning models, potentially enhancing their predictive capabilities.

A key advantage of complementing datasets with graph embeddings lies in their effectiveness in handling complex, interconnected data structures. Unlike conventional feature engineering methods, which may struggle to capture intricate relationships in the data, graph embeddings inherently encode such relationships through proximity in the embedding space. This makes them a well-suited candidate for tasks involving datasets with limited or poor-quality data, where traditional feature engineering approaches may fall short. Moreover, graph embeddings based on word2vec are less affected by missing values throughout the dataset. For instance, while traditional machine learning might discard features with a high percentage of missing values, word2vec-based models can integrate such features into the embedding seamlessly. Additionally, incorporating semantic information, such as classes and properties from a knowledge graph, into the embedding provides context to the machine learning model, giving the model a better understanding of the problem and potentially enhancing its performance.

To provide context and illustrate the relevance of the research, let's introduce a motivating example. Consider a hospital that relies on a tabular dataset containing hospital records, laboratory results, physical measurements, and demographic information. This conventional data structure may fall short in predicting complex hereditary diseases due to its inability to capture the intricate relationships and contextual nuances essential for accurate predictions.

Now, envision a knowledge graph constructed for the same patient population, with a highly expressive schema that includes not only patients and their diagnoses but also details the relationships between various entities, such as family connections, shared environmental factors, and treatment histories. For instance, understanding family relationships, such as ancestors or siblings, and shared living conditions can be crucial for predicting hereditary diseases or treatment responses. However, these critical relational and contextual details are typically absent in traditional tabular data.

The integration of graph embeddings with tabular data bridges this gap by incorporating the rich relational and structural information from a knowledge graph. This method provides an automated way of embedding the information found in the knowledge graph, eliminating the need for manually extracting relevant features from the knowledge graph. Graph embeddings can cover various aspects, such as a patient's medical conditions, the medical histories of their relatives, and the relationships between different diseases, assuming this information exists in the knowledge graph. For example, embeddings might reveal that patients with hypertension are often at risk of developing heart disease given the comorbity relationship they share in Figure

1. Graph embeddings could also enhance models designed for predicting Alzheimer's disease by incorporating family relationships of the patient. These examples are under the assumption that the tabular data does not include the relevant features to identify such relationships.

Therefore, applying graph embeddings enables a more complete and comprehensive data representation, ultimately enhancing the predictive power of machine learning models. This approach not only improves the ability to predict complex hereditary diseases and treatment outcomes but also highlights the broader applicability of knowledge graphs in enriching data representations across diverse domains.

Figure 1: Motivation Example Graph. Yellow nodes are schema and blue nodes are instances.

4 Related Work

In recent years, there has been a notable surge in the popularity of knowledge graphs, accompanied by a rapid increase of applications taking advantage of them. Given the computational expense associated with operations on large-scale knowledge graphs, researchers have increasingly turned their attention towards the development of knowledge graph embedding algorithms aimed at generating vectors that summarise the information found in the graph. Various families of embedding methodologies have emerged, each tailored to specific application contexts. The subsequent sections will include a classification of embedding methodologies followed by related work applying some of the aforementioned algorithms.

4.1 Translation Method Algorithms

Translation methods conceptualise link prediction as a geometric challenge, wherein entities and relations within a graph are represented within a unified vector space. The primary objective of these methods is to ensure that the translation of a relation r from a head entity h yields a tail entity t. Some examples are TransE [4], TransH [43], TransR [25], and TransD [21].

4.2 Matrix Factorisation Algorithms

Matrix factorisation algorithms have been devised to preserve essential graph properties, particularly those related to nodes, such as node similarity. Prominent examples within this category include Graph Laplacian Eigenmaps [3] [18] [35], Node Proximity Matrix Factorisation [1] [45] [8], and RESCAL [28].

4.3 Deep Learning Algorithms

Deep learning approaches have received considerable attention due to their application across diverse domains. These methodologies harness the power of deep learning to generate knowledge graph embeddings. They can be broadly categorised into two groups:

- **Random walk-based:** These algorithms typically entail two stages. Initially, a set of random walks is conducted for each node, followed by the conversion of these walks into sequences, which are subsequently fed into algorithms like word2vec. Notable representatives within this category include node2vec [17], RDF2Vec [34], and OWL2Vec* [9]. The embeddings derived from these methods are often better suited for downstream data mining tasks, including clustering, node similarity assessment, and node classification, compared to geometric solutions.
- **Non-random walked-based:** Approaches without random walks operate by considering the entire graph as input and apply deep learning models, such as AutoEncoders [42] [40] [7] and Graph Neural Networks [23] [36] [29] [6], to capture optimal neighbourhood representations. These solutions are versatile by being able to incorporate different types of edges and nodes, and offer a global representation of the graph. These advantages come at the cost of complexity, as, for example GNN, have a expensive requirements in computation and memory resources. [46]

4.4 Experimental Evaluations of KG Embeddings

Numerous machine learning solutions have been devised for Semantic Web challenges, yet their comparative evaluation has often been hindered by the absence of publicly available benchmark datasets. Addressing this gap, [33] curated benchmark datasets of varying sizes, drawn from existing Semantic Web repositories as well as from external machine learning problem domains associated with datasets within the Linked Open Data cloud.

Subsequently, these datasets were used in the study by Portisch et. al. [32], which conducted experiments to compare link prediction algorithms, such as Trans-E, Trans-R, Trans-H, DistMult [44], and RESCAL with embedding algorithms tailored for downstream data mining tasks, such as RDF2Vec and node2vec. The findings underscored the superior performance of algorithms customised for specific tasks, notably TransE for link prediction and RDF2Vec for data mining applications. However, notable exceptions emerged. For instance, RDF2Vec exhibited inferior performance compared to TransE and RESCAL in clustering tasks. The authors concluded that despite differing objectives, both research streams exhibited similarities. Furthermore, RDF2Vec appeared to encode similarity and relatedness within its embeddings, while TransE and its derivatives focused solely on similarity. Moreover, RDF2Vec demonstrated greater stability in scenarios where knowledge graphs exhibited noise and deviated weakly from their schema. In this work, the embedding algorithms were also tested on classification and regression tasks. However, only the embeddings were used for the ML tasks, they were not combined with the features from the original datasets.

Yu et. al. [47] used the Amsterdam Museums dataset from [33] whilst developing the Knowledge Embedding based Graph Convolutional Network which achieved state of the art metrics for entity classification and graph alignment.

More datasets from [33] featured in [37] as they introduced INK, Instance Neighbouring by using Knowledge a technique to learn binary feature-based representations, which are comprehensible to humans, for nodes of interest in a knowledge graph. The algorithm developed in this paper outperformed RDF2Vec in node classification tasks and supervised anomaly detection.

4.5 Dataset Enrichment with KGs

Both Taveekarn [39] and Vandewiele [41] developed tools to enrich datasets by associating entities with DBpedia and extracting relevant features. Their methods involve augmenting datasets with additional features derived from a knowledge graph. Similar to this thesis, their focus is on dataset enrichment; however, they implement feature selection over a knowledge graph and integrate these features into the dataset. For instance, they might incorporate the `income` property from a knowledge graph into a finance-based dataset. In contrast, our solution merges the original dataset with embeddings from the knowledge graph.

In summary, there has been related work involving the same algorithms as we have applied. Nonetheless, the end goal of each paper is different to what we want to achieve. Our objectives involve investigating whether it is possible to improve model predictive performance by enriching a dataset with a knowledge graph embedding, and, ultimately, understanding how the information and structure of a knowledge graph affects the embeddings' quality.

5 Background

In this section, the theoretical foundations are presented to understand the research work performed in the following sections. Consequently, there will be an introduction into knowledge graphs, their syntax, and a real-life example of knowledge graphs in the form of DBpedia. Following this, the concept of knowledge graph embeddings and related algorithms will be covered.

5.1 Knowledge Graph

A knowledge graph (KG) is a directed labelled graph that can be formed from information from a single source or combine information from multiple sources to represent some knowledge. Entities such as cities, companies, and institutions are extracted from the sources and connections are built between them. These connections represent a type of relationship between two entities.

This thesis uses the DBpedia KG which represents a collaborative endeavor aimed at extracting structured data from diverse Wikimedia projects, such as Wikipedia, Wiktionary, Wikibook, and Wikidata. DBpedia constitutes a communal effort to create an expansive repository of structured knowledge. This structured information forms an Open Knowledge Graph (OKG), readily accessible to users across the web.

Due to much looseness in the use of terms such as ontology, knowledge base, and knowledge graph it has proven to be difficult to provide an exact definition of a knowledge graph [14]. However, a reasonable interpretation is the following. The architecture of a knowledge graph can be separated into two parts. First, the ontology or knowledge base. An ontology is a formal conceptualisation that contains the definition, relations, and properties of entities. It is characterised by its semantic expressiveness which allows it to model knowledge in many forms. In most cases, the ontology of a KG generally represents the schema. However, there are some cases where an ontology is populated, that is, it contains a set of instances. The second element of a KG, the reasoning or entailment engine allows for the discovery of new knowledge through the automatic integration of additional sources. The inferred knowledge could be in the form of instances: populating a KG with more instances. Or by enriching the structure of a KG by identifying new relationships between entities or finding new classes and subclasses. The adoption of the reasoning engine is what converts a knowledge base into a knowledge graph.

Having given a general definition of a KG and its components, the following section will describe some of the conditions that must be fulfilled in a correct knowledge graph. [38]

First, all statements in a knowledge graph must be unambiguous. For that reason, KGs are labelled directed graphs because without direction or labels, it would be impossible to encode any meaning into the structure of a KG. Addtionally, the KG must be formed of unambiguous units. One example can be found in the use of Universal Resource Identifiers, URIs, in RDF graphs. URIs are used as a sort of global identifiers for entities, types and relations avoiding ambiguities in the graph. [38]

Second, a knowledge graph must use a limited set of relation types that in the context of the open-world knowledge graph, represent a set of essential relations and types that are true no matter the context. That is, a set of canonical relations and types that are always present. By limiting the set of relation types, it forces knowledge graph editors to build the structure of the graph in a durable and scalable manner.

Finally, KGs must include explicit provenance. A knowledge graph without any justification of where the information came from to create entities or relations cannot be considered a correct KG. By enforcing explicit provenance, it increases the credibility and trust of the users, and without it, KGs could not be used in scientific research for example.

Moving on from the formalities, definition and conditions of a correct knowledge graph, there are some advantages of KGs over traditional relation databases for example.

As mentioned previously, KGs facilitate the integration of multiple data sources and give a semantic understanding of the data. Both of these advantages can lead to a better insight of the data. Next, the nature of a graph-like structure allow KGs to be flexible and scalable when it comes to adding new sources or data to the knowledge base. Additionally, the use of graphs allow for linking data from multiple domains leading to a richer view of knowledge. Knowledge graphs can be useful when it comes to machine learning and AI applications. For example, graph analytics can be used to identify communities or links among entities. Or KGs could be applied to encode domain knowledge not available in the data to enhance models in classification and regression tasks. This thesis will delve deeper into this last insight.

The importance around the structure of a knowledge has been covered. However, what guidelines or framework can be followed to ensure the correctness of the structure will be addressed in the following sections, where the main frameworks of knowledge graphs will be introduced.

5.1.1 RDF

The Resource Description Framework [10], RDF, is a framework for representing information on the Semantinc Web. RDF follows a graph-based data model whose core construct is a set of triples with each triple consisting of a subject, predicate, and object. RDF graphs can be represented as a directed labelled graph where subject and object are nodes and the predication is an arc or relation. There are four different types of nodes in an RDF graph: IRIs, Literals, Blank nodes, and triple terms.

IRIs and Literals are known as resources, and they represent real-life concepts. Resources, synonymous with entities, can be anything from physical objects to documents and numbers. IRIs, Internationalized Resource Identifier, are a generalisation of URIs. They are strings that conform to the syntax defined in RFC 3987, used to associate an entity with a universal identifier. On the other hand, Literals are used for values such as strings, numbers, and dates. An RDF statement, is generally considered as a binary relationship where two resources, subject and object, are shown to be related through a predicate. Blank nodes do not identify specific resources and are used in specific RDF-syntaxes or to represent complex relationships between resources, for example N-ary relations. However, it is known to be good practice to avoid the use of blank nodes in implementations due to their anonymity making them dificult identify, and making simple entailment intractable. [19]

In conclusion, an RDF graph is a set of RDF triples where each triple (s, p, o) can be formed in the following manner:

- s is an IRI or a blank node
- p is an IRI
- o is IRI, a blank node, a literal, or a triple term

and a triple term is 3-tuple (s, p, o) which is recursively defined as follows:

- If s is an IRI or a blank node, p is an IRI, and o is an IRI, a blank node, or a literal, then (s, p, o) is a triple term.
- if s is an IRI or a blank node, p is an IRI, and o is a triple term, then (s, p, o) is a triple term

5.1.2 RDF-S

RDF Schema [11], RDF-S, provides a data modelling vocabulary for RDF graphs. It is a semantic expansion of RDF that provides a mechanism for defining groups of entities and defining the relationships between them. Up until now, RDF graphs were just formed of instance triples where relations are defined on an individual level. With the introduction of RDF-S, classes and properties and more structured semantics can be defined to enrich the semantic understanding of RDF data.

RDF Schema is defined by a system of classes and properties, where properties are described by the classes of resources they are related to. The main properties introduced by RDF-S are domain and range. In the case of domain of a property, it refers to the class or classes of resources to which the property applies as a subject. It specifies the types of resources that can be subjects of statements that use the specified property. On the other hand, the range of a property specifies to the class or classes of resources to which the property applies as an object.

RDF Schema also introduces the notion of classes and subclasses. Resources can be seperated into groups known as classes where members of the class are known as instances. Additionally with the help of a reasoner engine, if class C is a subclass of C', all instances of class C will automatically be inferred as members of class C'. So, in the example below, all instances of the class Author, will also be members of the class Person and will inheret any attributes of the Person class.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/schema#> .

ex:Person rdf:type rdfs:Class ;
          rdfs:label "Person" .

ex:Author rdf:type rdfs:Class ;
          rdfs:label "Author" ;
          rdfs:subClassOf ex:Person .

ex:Book rdf:type rdfs:Class ;
         rdfs:label "Book" .

ex:wrote rdf:type rdf:Property ;
            rdfs:label "wrote" ;
            rdfs:domain ex:Author ;
            rdfs:range ex:Book .
```

5.1.3 OWL

The Web Ontology Language [26], OWL, extends both RDF and RDF-S, and offers additional constructs for expressing ontologies, i.e., formal descriptions of concepts, relationships between concepts, and constraints on how these concepts relate to each other. One of the main objectives of the introduction of OWL is to make it easier for machines to automatically process and integrate information available on the Web.

OWL has three sublanguages: OWL Lite, OWL DL, and OWL Full. OWL Lite has limited use of the full capabilities of OWL, however, it still covers some cardinality restrictions, property restrictions and characteristics, equality and inequality classes. Please refer to Table 1 for some examples of the OWL Lite syntax.

Feature	Description
Class	Defines a group of individuals that share some properties.
rdfs:subClassOf	Creates class hierarchies by stating that one class is a subclass of another class.
rdf:Property	Represents relationships between individuals or from individuals to data values.
rdfs:subPropertyOf	Creates property hierarchies by stating that one property is a subproperty of another property.
rdfs:domain	Limits the individuals to which a property can be applied.
rdfs:range	Limits the values that a property may have.
Individual	Instances of classes; properties may be used to relate one individual to another.
equivalentClass	States that two classes are equivalent, meaning they have the same instances.
equivalentProperty	States that two properties are equivalent, meaning they relate individuals in the same way.
sameAs	States that two individuals are the same.
differentFrom	States that an individual is different from other individuals.
inverseOf	States that one property is the inverse of another property.
allValuesFrom	Restricts a property to have a local range restriction associated with it.
someValuesFrom	Requires at least one value for a property to be of a certain type.
disjointWith	Classes may be stated to be disjoint from each other.
unionOf, complementOf, intersectionOf	Boolean combinations of classes and restrictions are allowed, such as unionOf, complementOf, and intersectionOf.
minCardinality, maxCardinality, cardinality	Allows cardinality statements for arbitrary non-negative integers.
complex classes	Syntax extended to allow arbitrarily complex class descriptions, including enumerated classes, property restrictions, and Boolean combinations.

Table 1: OWL Lite Syntax

OWL DL and OWL Full share the same vocabulary, but OWL DL imposes additional constraints. Specifically, OWL DL requires type separation, meaning that a class cannot simultaneously be an individual or a property, and a property cannot simultaneously be an individual or a class.

Regardless of the version of OWL, they all extend RDF and RDF-S by also allowing for the incorporation of metadata in the graph allowing a more complete understanding of the KG.

As OWL extends both RDF and RDF-S introducing more complexity and expressiviness into KG, the need for a more advanced ontology reasoner emerges. There are many ontology reasoners available in the literature. However, HermiT [15] was chosen due to its advantage in expressiveness over other reasoners such as ELK [22]. The HermiT reasoner is an OWL (Web Ontology Language) reasoner designed for the efficient processing of ontologies. It is capable of performing various reasoning tasks such as consistency checking, classification, and querying of OWL ontologies. HermiT uses hypertableau calculus, which is a highly optimised algorithm for handling complex ontological structures. By applying these advanced techniques, HermiT can efficiently infer implicit knowledge from explicitly stated facts.

5.2 Embeddings

Embeddings are numerical represenatations that help Machine Learning (ML), and Artificial Intelligence (AI), models understand the real world. Embedding algorithms map semi-structured or unstructured, such as text, images, or graphs, into a vector space where similar or related entities can be found close to eachother exemplified by Figure 2. For instance, one can observe that all the fish related points are found in a group seperate from the rest. Or, for example, that all the social media terms are also in their separate cluster in the negative values along the y-axis . Embedding algorithms can also be viewed as dimensionality reduction algorithms, as they are given high dimensional data and output low dimension vectors that try to retain the semantic, syntactic, and numeric relationships within the original data. Some examples of embedding models are PCA [24], SVD [13], word2vec [27], BERT [12], and GPT [5] amongst others.

Figure 2: word2vec Embedding Vector Space [30]

5.3 Knowledge Graph Embeddings

Embeddings have become useful in a vast number of scenarios thanks to their ability to summarise semi-structured and unstructured information in vector of floating-point numbers. In the case of graph embeddings, there exist multiple type of algorithms, each tailored for specific tasks.

In this thesis, experiments are performed with link prediction algorithms, in the form of TransE, and deep learning algorithms like RDF2Vec and OWL2Vec*. This choice of algorithms is inspired by the results of [32], where link prediction algorithms were compared against deep learning algorithms. The outcome showcased that TransE was the best performing link prediction algorithm. As for deep learning algorithms, RDF2Vec proved to be the superior algorithm. Nonetheless, an extension of RDF2Vec, OWL2Vec* was considered for this thesis as it is designed to work with OWL KGs, which tend to have richer semantics than RDF KGs. And, thus, the resulting embeddings should be encoded with richer semantics.

5.3.1 TransE

TransE is an energy-based model for learning low-dimensioanal vector embeddings for entities and relations in a graph. In TransE, triples take the form of (h, l, t) where h is head entity, l is the relationship, and t is the tail entity. And relationships are considered as translations from a head entity to a tail. Therefore, if the graph contains the triple (h, l, t), then it should hold that the vector of entity t is approximately equal to the vector associated to entity h plus the vector of relationships l. Thus, TransE can be seen as an optimisation problem where the goal is to reduce the error between vectors $h + l$ and t for every triple in the graph. Please refer to Equation 1. Finally, the output of the TransE is an embedding for each entity and an embedding for each type of relationship found in the graph. Therefore, if the graph had n entities and m relationship types, the output would $n + m$ embeddings. The dimension of the embeddings is a hyperparameter of the model.

$$\text{TransE}(h, r, t) = \|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_1 \tag{1}$$

Algorithm 1 TransE [4]

Require: Training set $S = \{(h, \ell, t)\}$, entities and relations sets E and L, margin γ, embeddings dimension k

Initialise $\ell \leftarrow \text{uniform}\left(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}}\right)$ for each $\ell \in L$
$\ell \leftarrow \frac{\ell}{\|\ell\|}$ for each $\ell \in L$
$e \leftarrow \text{uniform}\left(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}}\right)$ for each entity $e \in E$
loop
  $e \leftarrow \frac{e}{\|e\|}$ for each entity $e \in E$
  $S_{\text{batch}} \leftarrow \text{sample}(S, b)$ // sample a minibatch of size b
  $T_{\text{batch}} \leftarrow \emptyset$ // initialise the set of pairs of triplets
  for $(h, \ell, t) \in S_{\text{batch}}$ **do**
    $(h', \ell, t') \leftarrow \text{sample}(S'_{(h,\ell,t)})$ // sample a corrupted triplet
    $T_{\text{batch}} \leftarrow T_{\text{batch}} \cup \{((h, \ell, t), (h', \ell, t'))\}$
  end for
  Update embeddings w.r.t. $\sum_{((h,\ell,t),(h',\ell,t'))\in T_{\text{batch}}} \nabla[\gamma + d(h + \ell, t) - d(h' + \ell, t')]_+$
end loop

Algorithm 13 goes as follows. First, the vectors for the entities, e, and relations, l are initialised to a random uniform distribution as suggested by [16]. At each iteration of the algorithm, the entity embeddings are normalised and a sample of size b triplets are selected for the training batch. For each triplet in this batch, a corrupted triplet is generated and added to the training batch. The embeddings are then updated to minimise the loss function for that batch. The algorithm is stopped based on its performance on a validation set.

5.3.2 RDF2Vec

RDF2Vec differs from TransE by only creating embeddings for each entity found in a graph. RDF2Vec preprocess an RDF graph in such a manner that it can be fed to a language, word2vec in this case. To achieve this, the RDF graph needs to be converted into a set of sentences that will be fed to the language model. This can be achieved with two mechanisms:

Graph Walks For a given graph $G = (V, E)$, walks of a depth d for each vertex (entity) V are generated using a bread-first algorithm.

Weisfeiler-Lehman Subtree RDF Graph Kernels: The Weisfeiler-Lehman graph kernel algorithm is adapted for RDF graphs through two key modifications. Firstly, RDF graphs feature directed edges, needing consideration of outgoing edges only in a vertex's neighborhood. Secondly, to address label variation across iterations while representing the same subtree, the authors introduce tracking of neighboring labels from the previous iteration using multisets. If the current iteration's multiset matches the previous one, the label from the previous iteration is reused.

Converting the RDF graph into sequences of tokens involves defining algorithm parameters, such as iterations and vertex subgraph depth, extracting paths of specified depth within each vertex's subgraph on the relabeled graph, and iteratively repeating this process until reaching the maximum iterations. The resulting set of sequences comprises the union of sequences for all vertices in each iteration.

Both of these methods contribute to the first part of the RDF2Vec algorithm, the document composition for training the language. Table 2 boasts some examples of the sequences that the word2vec model will be trained on.

Sentence
Hamburg → country → Germany → leader → Angela Merkel
Germany → leader → Angela Merkel → birthPlace → Hamburg
Hamburg → leader → Peter Tschentscher → residence → Hamburg

Table 2: RDF2Vec Examples

The model component of RDF2Vec is formed by word2vec, a computationally-efficient two-layer neural net model for learning word embeddings from raw text. The training process of the word2vec model comprises three main components: the vocabulary builder, the context builder, and the neural network.

First, the vocabulary builder extracts unique words from the input text, which can be sentences or, in the case of RDF2Vec, sequences from random walks over a graph. It scans through the input data to identify all unique words and compiles them into a corpus. This step involves tokenising the text, which means splitting it into individual words and filtering out any duplicates to form a comprehensive corpus of unique words.

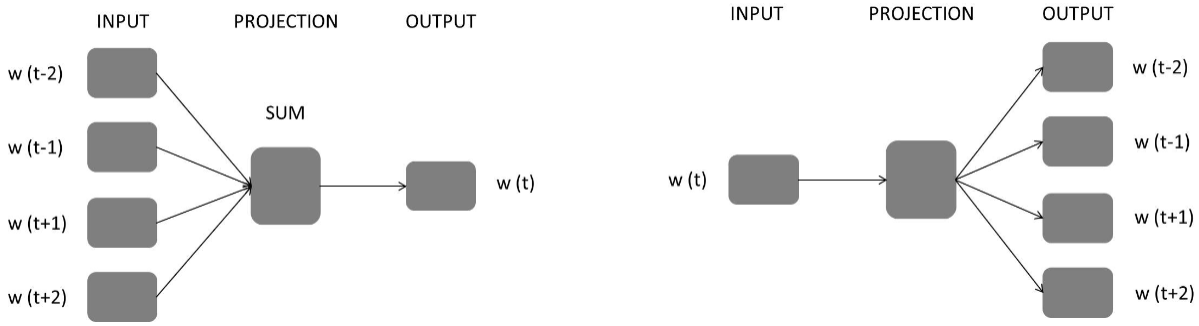
Next, the context builder generates word pairs based on a given context window, which helps the model learn the contextual relationships between words. For each word in a sentence, the

context builder considers a window of surrounding words. For example, with a context window of size 2, the word "spotted" in the sentence "The spotted dog" would form pairs with "The" and "dog". This process results in a set of word pairs that represent the local context in which each word appears, helping the model understand the relationships between words.

The core of the model is the neural network, a shallow, two-layer architecture. The input layer consists of neurons equal to the number of unique words in the vocabulary. Each input word is represented as a one-hot encoded vector, where only the index corresponding to that word is set to 1, and all other indices are set to 0. The hidden layer contains neurons equal to the desired dimensionality of the word embeddings. Therefore, if the desired dimensionality of the embeddings were 300, there would be 300 neurons. The size of this layer is a critical hyperparameter, as it determines the dimensionality of the resulting word vectors. The output layer, similar to the input layer, consists of neurons equal to the number of words in the vocabulary. The output of the neural network is a probability distribution over the vocabulary, indicating the likelihood of each word appearing given a word or some context depending on the training method. There are two ways of training word2vec:

Continous Bag Of Words The CBOW model is trained to predict a word given a context window, that is the words that surround it. Please refer to Figure 3a for an insight of its architecture.

Skip-Gram The Skip-Gram model is trained to predict the context words given a word. Please refer to Figure 3b for an insight of its architecture.



(a) CBOW Model

(b) Skip-Gram Model

Figure 3: word2vec Training Architectures [34]

5.3.3 OWL2Vec*

OWL2Vec* can be considered as an extension of RDF2Vec, where a word2vec model is trained on a corpus extracted from a knowledge graph with OWL statements. In this section, there will be more focus on the construction of the corpora as opposed to the training of the model. A corpus is extracted from the knowledge graph, converted from an OWL ontology to RDF using W3C mappings [31] and an OWL entailment reasoner, in our case the HermiT reasoner previously explained in Section 5.1.3. The corpus is formed by three documents: structure, lexical and combined documents. The first two aim to capture logical constructors and lexical information (labels and comments). The last preserves the correlation between entities (IRIs) and lexical information (words).

Structure Document: The structure document is composed by three types of sentences. The first type are random walks from a node in the graph. To perform the random walk, the RDF graph, G, is converted into a a directed single relation graph, G'. For each triple $\langle X, r, Y \rangle$ in G, X, r, Y are converted into nodes and two edges are added connecting them. This type corresponds to Ex1 and Ex2 found in Table 3.

The second type of sentences is similar except that they contain WL kernels which encode the structure of a subgraph. This corresponds to Ex3 in Table 3.

Finally, the third type of sentences are aimed at capturing the logical constructors. In OWL2Vec*, each ontology axiom is converted to a sequence following the OWL Manchester Syntax [20]. This type of sentence corresponds to Ex4 in Table 3. It is preferred over random walks over the projected RDF graph because it is shorter and avoids blank nodes.

Lexical Document: The lexical document includes two type of sentences. The first type are formed by taking sequences found in the structure document and translating the IRIs with their associated `rfds:label`. Ex5 and Ex6 from Table 3 showcase this transformation.

The second type of sequence are extracted from annotations in the graph using properties like `rdfs:comment`, `rdfs:seeAlso`, and other annotation properties. Ex7 in Table 3 is an example of this type of sentence.

Combined Document: OWL2Vec* enhances the extraction process by combining structural documents with entity annotations to preserve the relationship between entities (IRIs) and words in lexical information. Two strategies are developed to handle IRI sentences within the structural document. The first strategy involves randomly selecting an entity within an IRI sentence, retaining its IRI, and replacing other entities with their lowercase word tokens extracted from labels or IRI names. This strategy corresponds to Ex8 in Table 3. The second strategy involves traversing all entities in an IRI sentence, generating multiple combined sentences where each entity's IRI is preserved, and others are replaced with lowercase word tokens. This combined document aims to capture correlations between IRIs and words, facilitating the embedding of IRIs with word semantics and incorporating graph structure semantics into the word embeddings. While beneficial for certain contexts such as ontology-tailored word embedding models, it may introduce noise and negatively impact the word embeddings. Ex9 in Table 3 showcases an example of this second strategy.

Document	Example name and strategy	Sentence(s)
Structure	Ex1: random walk of depth tree starting from vc:FOOD-4001 (blonde beer) Ex2: random walk of depth four starting from vc:FOOD-4001 (blonde beer) Ex3: the walk in Ex2 with the WL sub-graph kernel enabled Ex4: the existential restriction of obo:FOODON_00002809 (edamame)	(vc:FOOD-4001, vc:hasNutrient, vc:VitaminC_100, vc:amountNutrient) (vc:FOOD-4001, rdf:type, vc:Beer, rdfs:subClassOf, vc:AlcoholicBeverages) (vc:FOOD-4001, rdf:type, kernel_id1_md5, rdfs:subClassOf, kernel_id2_md5) (obo:FOODON_00002809, subClassOf, obo:RO_0001000, some, obo:FOODON_03411347)
Literal	Ex5: replacing the IRIs in Ex1 by their labels or names Ex6: replacing the IRIs in Ex3 by their labels or names Ex7: the obo:IAO_0000115 (definition) annotation of obo:FOODON_00002809 (edamame)	(“blonde”, “beer”, “has”, “nutrient”, “vitamin”, “c”, “amount”, “nutrient”) (“blonde”, “beer”, “type”, kernel_id1_md5, “subclassof”, kernel_id2_md5) (“edamame”, “edamame”, “is”, “a”, “preparation”, “of”, “immature”, “soybean”, “in”, “their”, “pods” ...)
Combined	Ex8: keeping the IRI of one item in the Ex1 sentence (random strategy) Ex9: keeping the IRI of one item in the Ex1 sentence (traversing strategy)	(vc:FOOD-4001, “has”, “nutrient”, “vitamin”, “c”, “amount”, “nutrient”) (vc:FOOD-4001, “has”, “nutrient”, “vitamin”, “c”, “amount”, “nutrient”), . . . , (“blonde”, “beer”, “has”, “nutrient”, “vitamin”, “c”, vc:amountNutrient)

Table 3: Sentence examples that are extracted from the ontology fragments in Fig. 1

6 Methodology

This section outlines the process of using graph embeddings to enrich poor-quality datasets. Initially, the steps to obtain datasets containing the URIs of entities as an attribute are detailed. Subsequently, the construction of the appropriate KGs is described, by creating subsets of the DBpedia KG. To understand how the format of relevant data in the KG affects the quality of the embeddings, modifications to the graphs were made. The implementations of TransE and OWL2Vec* algorithms are introduced. Furthermore, modifications to the OWL2Vec* algorithm are explained to incorporate additional information from the KG that was not considered in the original implementation. Finally, the training and evaluation process of the embedding and machine learning algorithms is detailed.

6.1 Process

Figure 4 represents an overview of the steps taken to train machine learning models on enriched datasets. First, given a dataset containing URIs that link each instance to an entity in a KG, the URIs are extracted and used to query said KG. This query retrieves triples, concentrating on the URIs pertinent to the dataset. These triples are then integrated with the KG schema to create a subgraph centered on the relevant entities. Subsequently, this subgraph is processed by a graph embedding algorithm, resulting in a set of embeddings. The embeddings of interest are then selected and merged with the original dataset through an inner join on the URIs. Ultimately, the combined dataset is ready to be used for machine learning algorithms. The subsequent sections provide a detailed discussion of each step in this process.

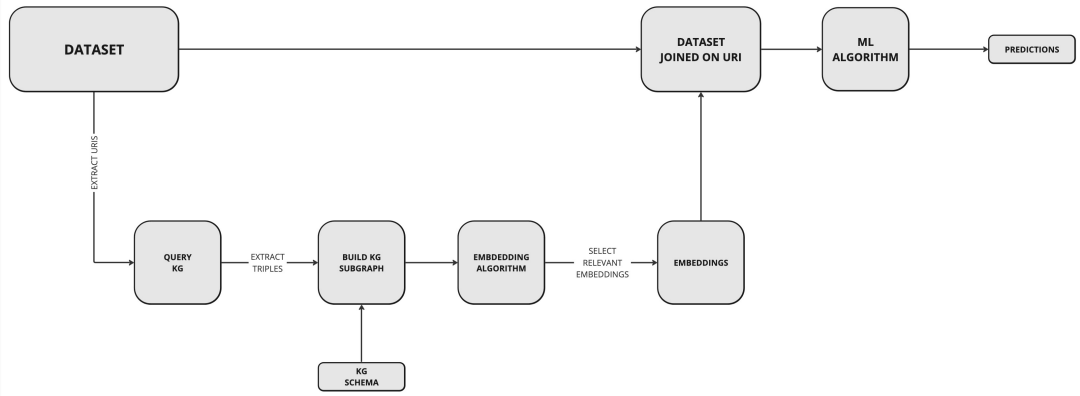


Figure 4: Process

6.2 Datasets

To combine a dataset with KG embeddings, there must be a unique identifier that links the instances in the dataset with the entities in the KG. If the dataset instances represent entities that are included in a KG, then the dataset can be enriched with graph embeddings. Nonetheless, the process of linking a dataset to a KG is costly and time-consuming.

Fortunately, Ristoski et. al. [33] took on this laborious task and produced a repository that includes a total of 22 datasets, some of which are classic machine learning datasets modified to include the URI of instances within the dataset. However, there was a specific criteria to follow, as not any dataset could be used.

Some datasets only included the label to predict and the URIs. However, for our evaluation, it is necessary to have some features to establish a baseline model for comparison. This allows us to compare the performance of models trained on the original datasets with those trained on

the enriched datasets. Additionally, the datasets must contain at least numeric features, and optionally, categorical features. This requirement is primarily due to the inclusion of regression tasks in our evaluation. Since the baseline model uses features from the original dataset, relying solely on categorical features for the baseline would be unfair as these features tend to perform worse in regression tasks. Out of the original 22 datasets, 4 were deemed acceptable for our use case.
The original datasets used were the following:

- **Forbes:** Contains a list of companies including several features of the companies, which was generated from the Forbes list of leading companies 2015. The target is to predict the company's market value as a classification and regression task.
  - **Size:** 1585 instances
  - **Features:** *Company*, *Industry*, *Country*, *Sales*, *Profits*, *Assets*, *Rank*, *URI*
  - **Label:** *Market Value*
- **AAUP:** Contains a list of American universities, including eight target variables describing the average salary of professors at the universities and the number of professor of different ranks. The target is to predict the type of instution which could be public, private, research university, liberal arts college.
  - **Size:** 960 instances
  - **Features:** *Average salary full professors, Average salary associate professors, Average salary assistant professors, Average salary all ranks, Average compensation full professor, Average compensation associate professors, Average compensation assistant professors, Average compensation all ranks, Number of full professors, Number of associate professors, Number of assistant professors, Number of instructors, Number of faculty all ranks, FICE, College name, State, URI*
  - **Label:** *Type*
- **Auto MPG:** Captures different characteristics of cars, and the target is to predict the fuel consumption (MPG) as a regression task. However, the target has also been discretised into "high", "medium" and "low", using equal frequency binning.
  - **Size:** 371 instances
  - **Features:** *Car, Cylinders, Displacement, Horsepower, Weight, Acceleration, Model, Origin, URI*
  - **Label:** *MPG*
- **Auto 93:** captures different characteristics of cars, and the target is to predict the price of the vehicles as a regression task. Again, the target has also been discretised into "high", "medium" and "low", using equal frequency binning.
  - **Size:** 93 instances
  - **Features:** *Car, Type, Minimum Price, Midrange Price, City MPG, Highway MPG, Air Bags, Drive train, Number of cylinders, Engine size, Horsepower, RPM, Engine revolutions per mile, Manual transmission available, Fuel tank capacity, Passenger capacity, Length, Wheelbase, Width, U-turn space, Rear seat room, Luggage capacity, Weight, Domestic, URI*
  - **Label:** *Maximum Price*

The main objective of this thesis is to enhance datasets of poor quality. If the datasets already contain all the attributes necessary for highly accurate predictions, there is little to no room for improvement by adding graph embeddings. Therefore, a decision was made to remove some of the original features from the datasets, creating space for potential enhancements through embeddings. This approach modifies the data available to simulate a use case scenario where enriching datasets with embeddings could be beneficial. Additionally, in some datasets, aggregation was applied to ensure that each URI belonged to only one instance. The following section details the modifications applied to each dataset:

- **Forbes:** Features such as *Company*, *Industry*, *Sales*, and *Assets* were removed to simulate a poor-quality dataset. The size of the dataset remained at 1585 instances.
- **AAUP:** Features like *College Name*, *State*, and *Type* were removed. Instances with null values in *Average salary associate professors* and *Average compensation assistant professors* were also removed. The instances were removed instead of the columns to maintain fairness by not excluding information about assistant professors while retaining features for full and associate professors. The size of the dataset was reduced to 933 instances.
- **Auto MPG:** No features were removed. However, instances where the URI was the brand of a car (e.g., Volkswagen) were removed because the information from the graph would not correspond to that of the actual car model. Aggregation was performed by car model, meaning instances sharing a model and URI had their numeric attributes aggregated by average and categorical attributes by mode. This reduced the dataset to 120 unique car models, not ideal for training a model.
- **Auto 93:** Features such as *Midrange Price* and *Minimum Price* were removed due to high correlation with the target variable. *Luggage capacity* and *Rear seat room* were removed due to containing NAs, as it was deemed that there were too few instances to justify their removal due to missing values. The dataset size remained at 93 instances.

6.3 OWL Graphs

Given that the datasets selected contained URIs to entities in DBpedia, this was selected as the appropriate KG for creating the embeddings. However, due to the extensive scope of DBpedia, computing embeddings for the entire graph was unfeasible. Moreover, even with ample resources and time, training an embedding algorithm on the entirety of DBpedia might not be advantageous, as it could potentially introduce noise into the entity embeddings due to the overwhelming volume of information. Consequently, to address these limitations, specialised subgraphs tailored to each dataset were constructed.

To ensure the correct functionality of OWL2Vec*, the subgraphs had to include the schema from the complete DBpedia graph. Therefore, this was taken as a starting point for each of the graphs constructed. Next, a query was performed to extract the entities and its properties from DBpedia. The number and types of properties extracted depends on the type of query. Full queries extract all properties and attributes excluding properties containing "wiki", for example `dbo:wikiPageWikiLink`, `dbo:wikiPageRevisionID`, `dbo:wikiPageId`, and image related properties. These properties just add noise to the embeddings reducing their quality, as they do not encode any relevant semantic information. Simple queries only extracted properties of type `rdf:type`, which linked the instances to the schema. Please refer to the Annex A.2 for the queries.

To explore how TransE and OWL2Vec* interpret graph information based on its presentation, several modifications have been made to the graphs:

- **No modifications:** The graph remains unchanged.

- **Less properties:** Selective feature extraction was applied to the Forbes and AAUP datasets, focusing on specific properties of interest and `rdf:type` to establish connections between instances and the schema. Notably, properties such as `dbo:numberOfEmployees`, `dbo:income`, and `dbo:assets` amongst others were included for Forbes, while `dbo:state`, `dbo:numberOfStudents`, `dbo:budget` were among those included for AAUP. These properties were included with the intuition that they could prove helpful for the downstream classification and regression tasks. Auto MPG and Auto 93 datasets lacked properties that warranted prioritisation.

Preliminary experiments showed that neither the plain graph embeddings nor the embeddings based on KGs with specific properties, improved the performance beyond the standard numeric models.

Therefore, a correlated feature linked to the target label was introduced into the graph in different ways, like as a property, attribute, and class. This adjustment helps understand what information the graph embeddings need and how it should be represented. Below we have the different degrees of correlation:

- **High correlation:** The correlated feature is equal to the label in 90% of the instances. The remaining 10% are incorrect values. Finally, 10% of the total instances are replaced with NA values to ensure that this information will not always be available.
- **Moderate correlation:** The correlated feature is equal to the label in 50% of the instances. The remaining 50% are incorrect values. Finally, 10% of the total instances are replaced with NA values.
- **Low correlation:** The correlated feature is equal to the label in 20% of the instances. The remaining 80% are incorrect values. Finally, 10% of the total instances are replaced with NA values.

The correlated feature can appear as:

- **Property:** The correlated feature appears as a property between all entities with the same value for this feature. For instance, all entities with the value "high" for this feature are connected with the property `ns2:high`. Figure 5 depicts this modification. The *Property* modification is expected to positively affect the performance of TransE because it is an algorithm designed for link prediction which focuses mainly on the properties between nodes.

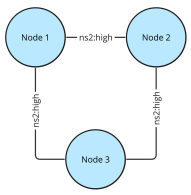


Figure 5: Property Modification

- **Class:** The correlated feature appears as a class inside the graph. Therefore, if HSBC had a "low" value for this feature, the graph would be modified so that the HSBC node would now be of type `ns2:lowCorrFeat`. Please refer to Figure 6. The *Class* modification is expected to positively affect the performance of OWL2Vec* because in [9], it performed extremely well in class membership and subsumption tasks. Thus, it is expected that it prioritises its focus on the classes of entities.

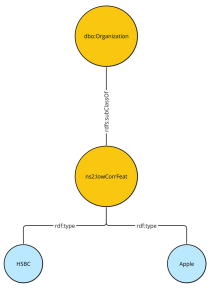


Figure 6: Class Modification

- **Attribute:** The correlated feature appears as a Literal property within the graph. For example, if the HSBC entity from the Forbes dataset had a "low" value for this correlated feature it would appear as Figure 7. The *Attribute* modification is introduced to discover whether it is equally effective as the *Class* modification.

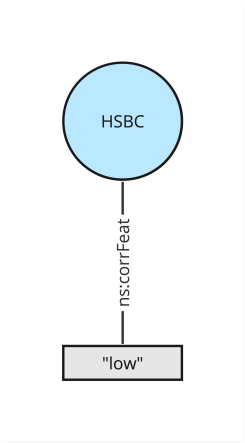


Figure 7: Attribute Modification

All degrees of correlation were considered for the *Class* and *Property* modifications. Only the high degree of correlation was considered as initial experiments proved the *Attribute* modification to be less significant than the *Class* and *Property* modifications.

Dataset	Query Type	Mods	Total Props	Dist Props
Forbes	Full	None	146987	708
		Property High Corr	944636	711
		Property Moderate Corr	779088	711
		Property Low Corr	769482	711
		Class High Corr	82600	709
		Class Moderate Corr	82600	709
		Class Low Corr	82600	709
		Attribute High Corr	-	-
	Simple	None	63268	22
		Property High Corr	926737	25
		Property Moderate Corr	761189	25
		Property Low Corr	751583	25
		Class High Corr	64701	23
		Class Moderate Corr	64701	23
		Class Low Corr	64701	23
		Attribute High Corr	64698	23
AAUP	Full	None	141599	578
		Property High Corr	342130	581
		Property Moderate Corr	310846	581
		Property Low Corr	306838	581
		Class High Corr	69542	579
		Class Moderate Corr	69542	579
		Class Low Corr	69542	579
		Attribute High Corr	-	-
	Simple	None	57752	22
		Property High Corr	331186	25
		Property Moderate Corr	299902	25
		Property Low Corr	295894	25
		Class High Corr	58598	23
		Class Moderate Corr	58598	23
		Class Low Corr	58598	23
		Attribute High Corr	58595	23

Table 4: Forbes and AAUP Graph Details

Dataset	Query Type	Mods	Total Props	Dist Props
Auto MPG	Full	None	49503	168
		Property High Corr	56751	171
		Property Moderate Corr	56823	171
		Property Low Corr	56743	171
		Class High Corr	52963	169
		Class Moderate Corr	52963	169
		Class Low Corr	52963	169
		Attribute High Corr	-	-
	Simple	None	36248	22
		Property High Corr	40150	25
		Property Moderate Corr	40222	25
		Property Low Corr	40142	25
		Class High Corr	36362	23
		Class Moderate Corr	36362	23
		Class Low Corr	36362	23
		Attribute High Corr	36359	23
Auto 93	Full	None	45593	113
		Property High Corr	50499	116
		Property Moderate Corr	50516	116
		Property Low Corr	50503	116
		Class High Corr	48251	113
		Class Moderate Corr	48251	113
		Class Low Corr	48251	113
		Attribute High Corr	-	-
	Simple	None	35261	22
		Property High Corr	37598	25
		Property Moderate Corr	37615	25
		Property Low Corr	37602	25
		Class High Corr	35350	23
		Class Moderate Corr	35350	23
		Class Low Corr	35350	23
		Attribute High Corr	35347	23

Table 5: Auto MPG and Auto 93 Graph Details

Tables 4 and 5 showcase the different variations of graphs created and the effect these modifications have on the number of properties found in the graph. For example, it can be seen that the *Property* modifications introduce a large number of properties in the graph, and increase the number of distinct properties by three, equal to the number of distinct values for the correlated feature. Meanwhile the *Class* and *Attribute* changes introduce a much lower number of properties into the graph.

6.4 Embedding Algorithms and Modifications

This section will discuss the implementations of TransE and OWL2Vec* employed along with our modification. Additionally, the configurations used to train both the OWL2Vec* and TransE algorithms are stated.

6.4.1 OWL2Vec* Original Implementation

OWL2Vec* was published in [9], extending the RDF2Vec algorithm by [34] to produce embeddings of OWL knowledge graphs. The code available can be found here. As OWL2Vec* is an extension of RD2Vec, it was preferred for comparison with TransE. OWL2Vec* comprises two primary steps. Initially, the algorithm involves document construction, which entails extracting information from the knowledge graph, converting it to sentences, and building a corpus . The second step involves training a word2vec model on the corpus derived from the KG.

6.4.2 OWL2Vec* Extended Implementation

Upon investigation, it was observed that the original OWL2Vec* algorithm overlooked the incorporation of literal values into its training corpus, thereby excluding integers, strings, or booleans which could provide relevant information to the ML models. Therefore, our implementation extends OWL2Vec* by incorporating such literals into the training corpus for the word2vec component. This allowed us to include features present in the graph but absent in the dataset. This modification also enabled the execution of experiments involving the *Attribute* modification.

In the process of adding the literal properties from the KG to the word2vec training corpus, it was discovered that many literals were of numeric nature, such as income, number of students, or number of employees. Given the inherent challenge word2vec faces in interpreting numerical values or the relationships between them, this properties were discretised into four categories before adding them to the training corpus. So, given a numeric property of type `x`, all the values of properties of type `x` were collected and discretised into four categories. As a proof of concept, our implementation has a fixed set of categories. However, the labels of the categories and the discretising criteria could be parameterised in future implementations, so the user could have liberty to decide which categories and how they should be seperated.

To give an example of this process, suppose that the property `dpo:income` is part of the Forbes graph. All the values for `dbo:income` would be collected and discretised into: low, medium, high, and very high. So, instead of the training corpus containing the sentence `HSBC income 100,000,000`. It would contain `HSBC income very high`.

This modification enhances the likelihood that the word2vec algorithm can effectively interpret a company's income. The code for this extension can be found in the Annex A.3.

The configuration for the OWL2Vec* model is as follows: a random walk depth of 4, as a lower walk depth did not include the literals in the training corpus; an embedding size of 300 dimensions, which is the default value for OWL2Vec*; and 15 epochs, which displayed an overall good loss, observe Figure 8. The training method for word2vec used is Skip Gram, the default method for OWL2Vec*, and the word2vec context window is set to 5. The rest of the configuration was left to the default values of OWL2Vec*.

The median time taken to calculate each set of embeddings was approximately 26 minutes with experiments such as those using Forbes data taking longer than those with Auto 93 data. The experiments were run on a MacBook Air equipped with an Apple M1 chip, featuring an 8-core CPU 8GB of RAM, and an integrated 7-core GPU.

6.4.3 TransE

The library PyKEEN [2] was employed to use the TransE algorithm. The configuration parameters for the TransE implementation were consciously selected to align with those used in OWL2Vec* for consistency in comparison. The parameters are as follows: The embedding dimension was set to 300 to match the embedding dimension used in OWL2Vec*. A batch size of 256 was chosen to balance between computational efficiency and training stability. The model

Figure 8: OWL2Vec* Loss Evolution

was trained for 15 epochs, with early stopping implemented using a patience parameter of 2 epochs and a tolerance of 0.01 based on evaluation loss. This strategy ensures that the training process halts if the improvement in the evaluation loss falls below the tolerance threshold for 2 consecutive epochs, thus preventing overfitting and reducing unnecessary computational expense. Figure 9 showcases the general loss evolution for the TransE model, reflecting its convergence behaviour over the training epochs. Note that not all runs reach 15 epochs.

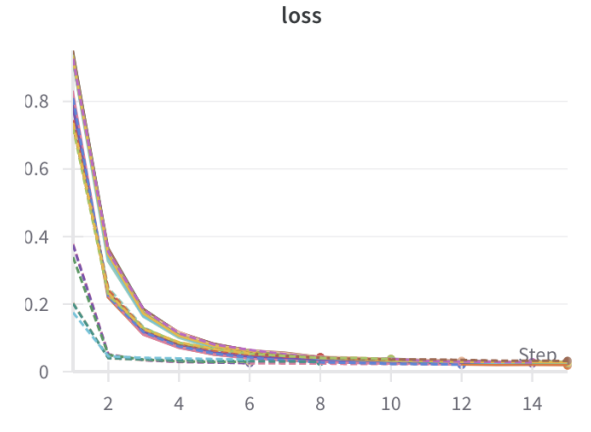


Figure 9: TransE Loss Evolution

The median time for calculating the emebddings with TransE was 45 seconds. However, the training time between the two algorithms cannot be compared due to the hardware used to run these experiments was different than OWL2Vec*, as TransE required a dedicated GPU and extensive RAM memory. The experiments were conducted on Google Colab Pro with an L4 runtime. This environment provided access to an NVIDIA L4 GPU with 22.5 GB of GPU memory. Depending on availability the RAM varied from 50 to 60GB.

6.5 Machine Learning Models

The machine learning models are designed to predict the labels for the original datasets as detailed in [33]. Specifically, the *Market Value* for Forbes, the *Type* of university for AAUP, the *MPG* for Auto MPG, and the *Maximum Price* for Auto 93. The baseline for each dataset is

a small neural network that takes as input the filtered numeric features of the dataset. In this process, categorical variables and highly correlated features are removed, the same features that were removed in Section 6.2.

The remaining models incorporate these numeric features along with a graph embedding derived from the combinations listed in Tables 4 and 5. An inner join is performed to merge the numeric features with the embeddings, excluding entities lacking either numeric features or embeddings. Standard Scaling is then applied to the numeric features to ensure their scale is similar to that of the embeddings. The embeddings themselves are not scaled to avoid altering their behaviour.

The combined dataset is divided into training (80%), test (15%), and validation (5%) sets. Given the limited number of instances, data augmentation is applied to the training set by duplicating it tenfold. No data augmentation is applied to the test and validation sets. While duplication is not the ideal form of data augmentation, it is better than generating synthetic instances that are similar to instances within the dataset.

There are simple data augmentation methods like SMOTE which works by generating new samples through linear interpolation between existing instances and their nearest neighbors of the same class. This approach is not suitable for embeddings because embeddings are complex, high-dimensional representations that capture non-linear relationships. Creating synthetic instances with linear interpolation can disrupt these relationships, altering the behavior and meaning of the embeddings.

More advanced methods like Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) are designed to handle high-dimensional data effectively. However, there's a risk: these methods may not fully grasp the complex relationships in the original embeddings, potentially leading to less accurate synthetic data. To avoid running this risk, these methods were not considered for data augmentation.

Classification and regression were selected as the machine learning tasks for evaluation for two reasons. First, these tasks align with the nature of the labels found in the datasets. Second, classification and regression are among the most popular machine learning tasks.

6.5.1 Classification Models

The neural network model implemented for the classification task is a sequential model created using TensorFlow. The architecture consists of multiple dense fully connected layers. The model begins with an input layer that has 128 neurons and uses the ReLU (Rectified Linear Unit) activation function. This layer includes an L2 kernel regularizer with a factor of 0.01 to prevent overfitting.

Following the input layer, a dropout layer with a rate of 0.6 is employed to further mitigate overfitting by randomly setting 60% of the input units to zero at each update during training. The subsequent layers are progressively smaller dense layers with 32, 16, and 12 neurons, respectively, each using the ReLU activation function and L2 kernel regularization with the same factor of 0.01.

The final layer is a dense layer with a number of neurons equal to the number of classes in the target variable and uses the softmax activation function. This setup allows the model to output a probability distribution over the classes for classification tasks.

The model is compiled using the Adam optimiser, which is an adaptive learning rate optimisation algorithm. The loss function used is sparse categorical cross-entropy, appropriate for multi-class classification problems. The model's performance is evaluated using F1-score as the metric.

To enhance training, early stopping is employed, which monitors the validation loss and stops

training if the loss does not improve for three consecutive epochs. This helps in preventing overfitting and reducing unnecessary training time. The model is trained on a batch size of 4 and a maximum number of 50 epochs, while also validating performance on a separate validation set.

To ensure that the experiment metrics are correct and stable, each experiment is run for 10 iterations obtaining the mean F1 score on the validation set.

6.5.2 Regression Models

Similar to the classification models, the regression models are sequential neural networks, incorporating regularisation techniques to avoid overfitting. The architecture consists of an input layer, two hidden layers, and an output layer. Each hidden layer using a ReLU activation function with 64 and 32 neurons, respectively. L2 regularisation has been applied with a factor of 0.01. Additionally, dropout layers with a dropout rate of 0.2 are inserted after each hidden layer. The model is compiled with the Adam optimiser, initialized with a learning rate of 0.001. The loss function chosen for this regression task is MSE (Mean Squared Error). Finally, it is trained with a batch size of 16 for a maximum of 50 epochs with an early stopping of 10 epochs patience.

Each experiment was run 10 iterations obatining the mean RMSE (Root Mean Squared Error) on the validation set.

6.6 Clustering

To perform the clustering analysis, embeddings generated by the TransE and OWL2Vec* were subjected to dimensionality reduction using the t-SNE algorithm, which condensed the high-dimensional data into a two-dimensional space. The resulting two-dimensional representations were visualised as a scatterplot, with each data point colour-coded according to its corresponding categorical label from the dataset.

6.7 Ethical Implications

An important topic to address is the ethical implications of using embedding algorithms. Similar to conventional machine learning and AI models, the bias and political correctness of our models are largely determined by the data on which they are trained. If the knowledge graph used to generate the embeddings or the dataset used to train the ML models contain unfair bias, this will be reflected in the embeddings and the final model.

In this thesis, the potential bias in our embeddings will primarily derive from the quality of the data within the DBpedia graph and the datasets from [33]. Given the specific datasets used for this research, predicting the market value of Forbes companies, the MPG and price of cars, and the type of college, it appears unlikely that there are evident factors that could introduce unfair bias into the results. Notably, none of these datasets involve features related to gender, age, or race, which are common sources of bias in predictive modeling.

7 Results

This section presents the results of the most relevant experiments along with a brief discussion. For a full view of the experiments, please refer to the Annex A.4. It begins with a comparison of the original OWL2Vec* implementation and our extended version on the classification task. Then, the performance of our extended OWL2Vec* version is compared with TransE on multiple KG configurations, including comparisons against the baseline. This is followed by a comparison of the algorithms on the regression task. Each experiment was run 10 times, calculating the average of the appropriate metric for each task. Finally, the clustering section is introduced to support the conclusions drawn from the classification task.

7.1 OWL2Vec* Implementation Comparison

Table 6 represents the difference in performance for classification tasks of the original implementation of OWL2Vec* compared to our extended version, including the discretised value of literals. In preliminary experiments, classification showcased less overfitting and more reliable results than the regression task, so it was chosen to compare the two versions of OWL2Vec*.

Only the full graphs with no modifications were considered because the simple graphs do not include literal values. Our implementation is an extension of the original OWL2Vec* because it includes literals into the embeddings. Therefore, if the simple graphs have no literals, both implementations of OWL2Vec* perform equally.

As can be seen in Table 6, the extended version performed equally or better in all cases. This slight difference in performance could stem from adding literals to the graph embeddings, potentially including features that would not be considered in the original implementation. Nonetheless, the extended version was used for the subsequent experiments.

Dataset	Query Type	Implementation	F1 Score	Baseline
Forbes	Full	Original	0.77	0.79
		Extended	0.78	
Auto 93	Full	Original	0.69	0.79
		Extended	0.77	
Auto MPG	Full	Original	0.79	0.76
		Extended	0.81	
AAUP	Full	Original	0.78	0.80
		Extended	0.78	

Table 6: OWL2Vec* Implementation Comparison

Despite, our implementation performing better slightly better than the original OWL2Vec* algorithm, it did not outperform the baseline in all cases. This could be due to a lack of relevant information in the DBpedia KGs. Thus, in the following sections, modified KGs from Section 6.3 are considered to evaluate the efficacy of the embedding algorithms to encode relevant features.

7.2 Classification Models

Table 7 includes a summary of all the experiments for the classification task with OWL2Vec*. The experiments were chosen based on performing a comparison of the Simple graphs with Full ones. Then, a comparison between the highly correlated graph modifications, to see which modification was interpreted best by OWL2Vec*. Finally, the low correlated version of the best modification, in this case the class modification, was chosen to introduce a slightly more realistic

use case scenario where the correlated variable shares approximately 20% of its values with the target.

As for the results, it is clear that, in most cases, the high class correlated modification does improve considerably over the baseline which is using only numeric attributes available in the datasets as described in 6.2. The results for the Auto 93 and Auto MPG are to be taken with a pinch of salt as they contain very few instances, 93 and 120, respectively. The lack of instances probably lead to the model overfitting on the training set. In the case of Forbes, it definitely seems that the modified graphs achieve a better performance than the baseline, meaning that if the KG contains useful information like a correlated features, graph embeddings could be used to enhance the performance of ML models in classification tasks. Additionally, it seems that OWL2Vec* can interpret relevant information in all formats considered, as a class, a property, and an attribute. Nonetheless, the correlated feature is definitely best understood when present as a class in the KG.

However, the conditions are not ideal, it seems that there needs to be very highly correlated features in the graph for embeddings to be worth while.

Dataset	Query Type	Mods	F1 Score	Baseline
Forbes	Simple	None	0.79	0.79
		Property High Corr	0.85	
		Attribute High Corr	0.83	
		Class Low Corr	0.81	
		Class High Corr	0.93	
	Full	None	0.78	
		Property High Corr	0.84	
		Class Low Corr	0.82	
		Class High Corr	0.92	
Auto 93	Simple	None	0.82	0.79
		Property High Corr	0.73	
		Attribute High Corr	0.71	
		Class Low Corr	0.38	
		Class High Corr	0.57	
	Full	None	0.76	
		Property High Corr	0.52	
		Class Low Corr	0.40	
		Class High Corr	0.52	
Auto MPG	Simple	None	0.62	0.76
		Property High Corr	0.64	
		Attribute High Corr	0.70	
		Class Low Corr	0.76	
		Class High Corr	0.90	
	Full	None	0.81	
		Property High Corr	0.75	
		Class Low Corr	0.76	
		Class High Corr	0.89	
AAUP	Simple	None	0.80	0.80
		Property High Corr	0.76	
		Attribute High Corr	0.82	
		Class Low Corr	0.79	
		Class High Corr	0.89	
	Full	None	0.78	
		Property High Corr	0.80	
		Class Low Corr	0.79	
		Class High Corr	0.88	

Table 7: OWL2Vec* Classification Results

Table 8 contains the results for the TransE embeddings on the simple query graphs. The full graphs were not considered as the TransE algorithm only focuses on the relationship between nodes, and it does not include the values of literals. Much like the OWL2Vec* comparison, this table contains a comparison of the plain graph embeddings with those of the highly correlated modifications, and the low correlated version of the best performing modification, in this case the property one.

Similarly to OWL2Vec*, it seems that the plain graph embeddings offer no improvement on the baseline. However, the highly correlated property modification does considerably improve the model's performance. And in the case of Forbes and AAUP, the low correlated property modification seems to perform slightly better than the baseline. Nonetheless, it is again a

Dataset	Query Type	Mod	F1 Score	Baseline
Forbes	Simple	None	0.76	0.79
		Property Low Corr	0.80	
		Property High Corr	0.89	
		Class High Corr	0.77	
		Attribute High Corr	0.75	
Auto 93	Simple	None	0.69	0.79
		Property Low Corr	0.46	
		Property High Corr	0.81	
		Class High Corr	0.68	
		Attribute High Corr	0.67	
Auto MPG	Simple	None	0.67	0.77
		Property Low Corr	0.75	
		Property High Corr	0.97	
		Class High Corr	0.80	
		Attribute High Corr	0.74	
AAUP	Simple	None	0.76	0.80
		Property Low Corr	0.81	
		Property High Corr	0.91	
		Class High Corr	0.78	
		Attribute High Corr	0.79	

Table 8: TransE Classification Results

case of the graphs having to include specific information to be able to offer an improvement in performance. Furthermore, the property modifications require a much larger number of properties and introduce more redundancy than the class modification. Please refer to Tables 4 and 5 and observe the large difference in total properties. It is highly unlikely that someone would design a KG like this in a real life situation because it is simply inefficient.

7.3 Regression Models

Tables 9 and 10 present an analysis of the performance of OWL2Vec* and TransE embeddings in regression tasks. The AAUP dataset was excluded from these experiments as its label is non-numerical, unlike the other datasets. The selection criteria for these experiments mirrored those used in the classification experiments.

The regression results on the Forbes dataset indicate that neither OWL2Vec* nor TransE embeddings contributed to an improvement in model performance, even in the presence of highly correlated features within the graph. In contrast, the embeddings appeared to significantly enhance the regression models for the Auto 93 and Auto MPG datasets. Notably, even the plain graph without modifications outperformed the baseline in these cases. However, due to the limited number of instances in both the Auto 93 and Auto MPG datasets, it remains uncertain whether the embeddings genuinely enhance the models. Given the classification results for these datasets, it is prudent to approach these findings with caution.

The observed performance disparity suggests that while graph embeddings can offer some benefits, further investigations with larger datasets and diverse contexts are necessary to draw more definitive conclusions about the effectiveness of these embedding techniques in regression tasks.

Dataset	Query Type	Mod	RMSE	MAE	Baseline RMSE	Baseline MAE
Forbes	Simple	None	63.71	45.71	57.23	45.70
		Class High Corr	59.47	42.51		
		Attribute High Corr	60.36	44.79		
		Property High Corr	61.55	45.44		
	Full	None	62.19	46.33		
		Class High Corr	58.21	41.87		
		Property High Corr	63.70	45.47		
Auto 93	Simple	None	10.08	7.93	29.68	24.84
		Class High Corr	12.28	10.85		
		Attribute High Corr	8.29	6.33		
		Property High Corr	10.81	8.73		
	Full	None	12.06	9.23		
		Class High Corr	9.51	7.92		
		Property High Corr	9.77	7.47		
Auto MPG	Simple	None	13.21	10.59	34.79	28.47
		Class High Corr	10.44	8.23		
		Attribute High Corr	13.30	9.93		
		Property High Corr	14.04	10.80		
	Full	None	15.87	12.14		
		Class High Corr	12.93	9.96		
		Property High Corr	15.53	11.66		

Table 9: OWL2Vec* Regression Results

Dataset	Query Type	Mod	RMSE	MAE	Baseline RMSE	Baseline MAE
Forbes	Simple	None	75.26	55.90	57.23	45.70
		Property Low Corr	83.03	59.73		
		Property High Corr	61.01	44.60		
		Class High Corr	70.11	51.30		
		Attribute High Corr	70.23	52.40		
Auto 93	Simple	None	12.29	9.55	29.68	24.84
		Property Low Corr	13.11	11.29		
		Property High Corr	10.99	9.27		
		Class High Corr	12.53	10.42		
		Attribute High Corr	11.55	8.66		
Auto MPG	Simple	None	13.55	9.97	34.79	28.47
		Property Low Corr	11.07	7.57		
		Property High Corr	11.08	9.06		
		Class High Corr	13.30	10.54		
		Attribute High Corr	9.33	7.03		

Table 10: TransE Regression Results

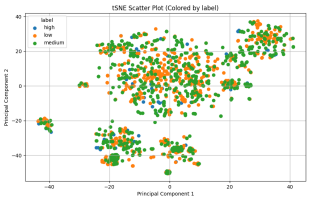
7.4 Clustering

Figures 10 and 11 showcase the clustering scatterplot of the Forbes embeddings. They corroborate the results found in Sections 7.2 and 7.3, particularly the classifications results. For example, it is notable that the TransE embeddings in Figure 11 struggle to learn the correlated

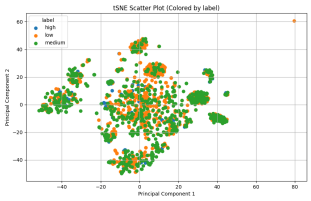
feature when it appears as a class or an attribute within the graph. On the other hand, TransE excels in the case of the property modification.

Furthermore, for OWL2Vec* embeddings, the class modification performs best and produces the best separation amongst the different classes of the Forbes dataset. Nonetheless, it is still able to detect the correlated feature when it is presented as an attribute of the entities or as a property of entities sharing the same value for the correlated feature.

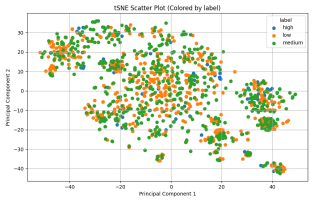
From these plots, it appears that OWL2Vec* exhibits greater versatility than TransE in comprehending relevant features, regardless of their form within the graph. However, each algorithm demonstrates distinct preferences: OWL2Vec* performs optimally when the knowledge graphs possess an expressive schema, while TransE excels in focusing on the relationships between entities.



(a) Simple Property High Correlation



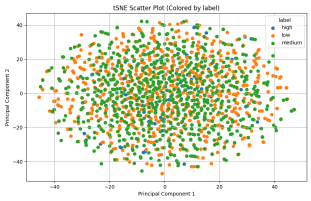
(b) Simple Class High Correlation



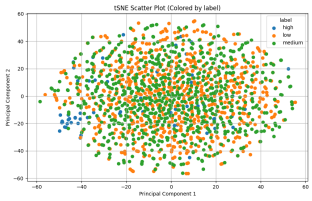
(c) Simple Attribute High Correlation

Figure 10: Forbes OWL2Vec* Clustering

(a) Simple Property High Correlation



(b) Simple Class High Correlation



(c) Simple Attribute High Correlation

Figure 11: Forbes TransE Clustering

8 Conclusion

The primary objective of this research was to evaluate the effectiveness of graph embeddings for enhancing the performance of ML models on poor quality datasets. In particular, we evaluated whether OWL2Vec* and TransE embeddings could improve the performance of classification and regression models. The task was to predict labels associated with datasets related to a KG. Contrary to our expectations, the embeddings produced by both OWL2Vec* and TransE when trained on the DBpedia KG failed to beat the baseline ML models.

As a secondary objective, we set out to assess whether our extension of OWL2Vec* outperformed the original version in downstream classification tasks. We concluded that our implementation performed equally or better in all cases. Therefore, our implementation of OWL2Vec* was preferred for the remainder of experiments. However, it must be noted, to be able to determine superiority of one implementation over another, more experiments should be performed on more datasets and a variety of tasks such as link prediction, node classification, class subsumption, and node similarity.

As for our third obejctive, to further understand the behaviour of the graph embeddings algorithm depending on the structure of the graph, a series of simulations were conducted. These simulations incorporated different degrees of correlated features within the graph, represented as attributes, properties, and classes. Our findings indicate that OWL2Vec* exhibits superior performance in classification and clustering tasks when a correlated feature is presented as a class within the graph. Additionally, OWL2Vec* demonstrated the capability to interpret highly correlated features when these are represented as attributes, while struggling when the features are presented as properties. This suggests that OWL2Vec* embeddings could be particularly beneficial in scenarios involving knowledge graphs with highly expressive schemas, where the granularity of entity types is high. For instance, if the hypothetical example in Section 3, is extended to include detailed classes for medical patients, such as diabetic patients, hypertensive patients, and heart disease patients. Another pertinent example could involve a financial knowledge graph with classes representing different types of transactions, like fraudulent transactions, legitimate transactions, and disputed transactions.

Further analysis revealed that TransE outperforms OWL2Vec* in interpreting properties, challenging the notion that a single embedding algorithm could serve as an all-round solution for capturing attributes, relationships and semantic information from the graph. A potential strategy to take advantage of the strengths of both algorithms involves using OWL2Vec* to capture the KG schema, entity classes, and literal values, while employing TransE to capture inter-entity relationships. Subsequently, combining both OWL2Vec* and TransE embeddings via concatenation could exploit the complementary strengths of each algorithm. So, in the case of our motivating example, OWL2Vec* would be more adept for capturing semantic information such as what is a disease and the different types of diseases. TransE could be better at representing relationships between entities, such as, relatives, comorbidities, and relationships between patients and diseases.

It is crucial to note that graph embeddings are redundant if the KG does not provide any additional information to the datasets. For example, if the KG only contains features that already exist in the datasets or if the prediction target is already embedded within the graph.

While graph embeddings can enhance machine learning models under certain conditions, these scenarios are highly specific and typically require control over the knowledge graph's design. Consequently, the practical application of graph embeddings to improve model performance is limited. In most instances, feature selection over the graph to enrich the dataset, as proposed by [41] and [39], may prove more practical, despite being more laborious, time-consuming, and requiring expert domain knowledge.

An intriguing future direction would be to replace the word2vec component of OWL2Vec* with a more advanced Large Language Model (LLM) such as BERT or GPT. Numerous studies have demonstrated that these models surpass word2vec in a variety of downstream tasks. Given the larger context window, the integration of attention mechanisms, and other advanced features, we hypothesise that embeddings generated by these LLMs would exhibit superior quality compared to those produced by word2vec. This avenue was considered for exploration in this thesis, however, limitations in resources and, ultimately, time made it unfeasible.

Other future work, could involve using a dataset and graph that epitomise the ideal scenario, akin to our motivating example.

In conclusion, while the current study highlights the potential and limitations of graph embeddings in enhancing machine learning models, it underscores the necessity for carefully designed knowledge graphs and the potential application of advanced language models in generating high-quality embeddings. Future explorations in this direction could yield significant advancements in the field of graph-based machine learning.

8.1 Personal Reflection

This thesis originates from an idea I had while reading Jan Portisch, Nicolas Heist, and Heiko Paulheim's paper [32], where they used graph embeddings to perform machine learning tasks. Although the graph embeddings did not achieve outstanding performance, they showed some promise in classification and regression tasks.

Inspired by this novel, at least to me, application of graph embeddings, I thought it could be interesting to explore whether graph embeddings could complement existing datasets to provide better solutions for machine learning tasks, such as classification and regression.

This thesis has allowed me to expand and apply the knowledge, concepts, and techniques I learned in various courses during my Master's in Data Science at UPC. For example, the Semantic Data Management (SDM) course introduced the topic of knowledge graphs and graph embeddings, sparking my curiosity to learn more about them and their applications. The Mining Unstructured Data (MUD) course taught me about NLP techniques, particularly large language models (LLMs). The combination of SDM and MUD enabled me to quickly understand the RDF2Vec and OWL2Vec* embedding algorithms and how they could be modified to potentially extract extra performance in machine learning tasks. The Machine Learning (ML) course taught me about neural networks, which I used for the classification and regression models. Finally, subjects such as Complex and Social Networks (CSN) further increased my curiosity about graphs and taught me how to perform rigorous experiments.

Overall, this thesis has allowed me to combine the knowledge and techniques acquired from both aspects of this Master's in Data Science: data analysis and data management.

References

[1] Amr Ahmed et al. "Distributed large-scale natural graph factorization". In: *Proceedings of the 22nd international conference on World Wide Web*. 2013, pp. 37–48.

[2] Mehdi Ali et al. "PyKEEN 1.0: a python library for training and evaluating knowledge graph embeddings". In: *Journal of Machine Learning Research* 22.82 (2021), pp. 1–6.

[3] William N Anderson Jr and Thomas D Morley. "Eigenvalues of the Laplacian of a graph". In: *Linear and multilinear algebra* 18.2 (1985), pp. 141–145.

[4] Antoine Bordes et al. "Translating embeddings for modeling multi-relational data". In: *Advances in neural information processing systems* 26 (2013).

[5] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[6] Joan Bruna et al. "Spectral networks and locally connected networks on graphs". In: *arXiv preprint arXiv:1312.6203* (2013).

[7] Shaosheng Cao, Wei Lu, and Qiongkai Xu. "Deep neural networks for learning graph representations". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[8] Shaosheng Cao, Wei Lu, and Qiongkai Xu. "Grarep: Learning graph representations with global structural information". In: *Proceedings of the 24th ACM international on conference on information and knowledge management*. 2015, pp. 891–900.

[9] Jiaoyan Chen et al. "Owl2vec*: Embedding of owl ontologies". In: *Machine Learning* 110.7 (2021), pp. 1813–1845.

[10] World Wide Web Consortium et al. "RDF 1.1 concepts and abstract syntax". In: (2014).

[11] Brickley Dan. "RDF vocabulary description language 1.0: RDF schema". In: *http://www.w3. org/TR/rdf-schema/* (2004).

[12] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[13] Carl Eckart and Gale Young. "The approximation of one matrix by another of lower rank". In: *Psychometrika* 1.3 (1936), pp. 211–218.

[14] Lisa Ehrlinger and Wolfram Wöß. "Towards a definition of knowledge graphs." In: *SEMANTiCS (Posters, Demos, SuCCESS)* 48.1-4 (2016), p. 2.

[15] Birte Glimm et al. "HermiT: an OWL 2 reasoner". In: *Journal of automated reasoning* 53 (2014), pp. 245–269.

[16] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[17] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864.

[18] Xiaofei He and Partha Niyogi. "Locality preserving projections". In: *Advances in neural information processing systems* 16 (2003).

[19] Aidan Hogan et al. "Everything you always wanted to know about blank nodes". In: *Journal of Web Semantics* 27 (2014), pp. 42–69.

[20] Matthew Horridge and Peter F Patel-Schneider. "OWL 2 web ontology language manchester syntax". In: *W3C Working Group Note* (2009).

[21] Guoliang Ji et al. "Knowledge graph embedding via dynamic mapping matrix". In: *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*. 2015, pp. 687–696.

[22] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. "ELK reasoner: architecture and evaluation." In: *ORE*. Citeseer. 2012.

[23] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[24] LIII KPFRS. "On lines and planes of closest fit to systems of points in space". In: *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (SIGMOD)*. 1901, p. 19.

[25] Yankai Lin et al. "Learning entity and relation embeddings for knowledge graph completion". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 29. 1. 2015.

[26] Deborah L McGuinness, Frank Van Harmelen, et al. "OWL web ontology language overview". In: *W3C recommendation* 10.10 (2004), p. 2004.

[27] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[28] Maximilian Nickel, Volker Tresp, Hans-Peter Kriegel, et al. "A three-way model for collective learning on multi-relational data." In: *Icml*. Vol. 11. 10.5555. 2011, pp. 3104482–3104584.

[29] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. "Learning convolutional neural networks for graphs". In: *International conference on machine learning*. PMLR. 2016, pp. 2014–2023.

[30] OpenClassrooms. *Discover the Power of Word Embeddings*. `https://openclassrooms.com/en/courses/6532301-introduction-to-natural-language-processing/8082110-discover-the-power-of-word-embeddings`. Accessed: 2024-05-29. 2023.

[31] Peter F. Patel-Schneider and Boris Motik. *OWL 2 Web Ontology Language Mapping to RDF Graphs (Second Edition)*. `https://www.w3.org/TR/owl2-mapping-to-rdf/`. 2012.

[32] Jan Portisch, Nicolas Heist, and Heiko Paulheim. "Knowledge graph embedding for data mining vs. knowledge graph embedding for link prediction–two sides of the same coin?" In: *Semantic Web* 13.3 (2022), pp. 399–422.

[33] Petar Ristoski, Gerben Klaas Dirk De Vries, and Heiko Paulheim. "A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web". In: *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II 15*. Springer. 2016, pp. 186–194.

[34] Petar Ristoski and Heiko Paulheim. "Rdf2vec: Rdf graph embeddings for data mining". In: *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I 15*. Springer. 2016, pp. 498–514.

[35] Sam T Roweis and Lawrence K Saul. "Nonlinear dimensionality reduction by locally linear embedding". In: *science* 290.5500 (2000), pp. 2323–2326.

[36] Baoxu Shi and Tim Weninger. "Proje: Embedding projection for knowledge graph completion". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[37] Bram Steenwinckel et al. "INK: knowledge graph embeddings for node classification". In: *Data Mining and Knowledge Discovery* 36.2 (2022), pp. 620–667.

[38] Frans N Stokman and Pieter H de Vries. "Structuring knowledge in a graph". In: *Human-Computer Interaction: Psychonomic Aspects*. Springer, 1988, pp. 186–206.

[39] Waran Taveekarn et al. "Data++: An automated tool for intelligent data augmentation using wikidata". In: *2019 16th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. IEEE. 2019, pp. 91–96.

[40] Fei Tian et al. "Learning deep representations for graph clustering". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 28. 1. 2014.

[41] Gilles Vandewiele. "Enhancing white-box machine learning processes by incorporating semantic background knowledge". In: *The Semantic Web: 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28–June 1, 2017, Proceedings, Part II 14*. Springer. 2017, pp. 267–278.

[42] Daixin Wang, Peng Cui, and Wenwu Zhu. "Structural deep network embedding". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 1225–1234.

[43] Zhen Wang et al. "Knowledge graph embedding by translating on hyperplanes". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 28. 1. 2014.

[44] Bishan Yang et al. "Embedding entities and relations for learning and inference in knowledge bases". In: *arXiv preprint arXiv:1412.6575* (2014).

[45] Cheng Yang et al. "Network representation learning with rich text information." In: *IJCAI*. Vol. 2015. 2015, pp. 2111–2117.

[46] Rex Ying et al. "Graph convolutional neural networks for web-scale recommender systems". In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 974–983.

[47] Donghan Yu et al. "Knowledge embedding based graph convolutional network". In: *Proceedings of the web conference 2021*. 2021, pp. 1619–1628.

A Annex

A.1 Dataset Descriptive Analysis

Forbes Dataset

Missing Values

Column	Missing Values
id	0
Company	0
Industry	0
Country	0
Sales	0
Profits	0
Assets	0
Rank	0

Top Attributes Most Correlated with 'label_num'

Attribute	Correlation
Profits	0.788
Sales	0.651
Assets	0.379
Rank	-0.541

AAUP Dataset

Missing Values

Column	Missing Values
uri	0
Average_salary_full_professors	47
Average_salary_associate_professors	25
Average_salary_assistant_professors	16
Average_salary_all_ranks	0
Average_compensation_full_professor	47
Average_compensation_associate_professors	25
Average_compensation_assistant_professors	16
Average_compensation_all_ranks	0
Number_of_full_professors	0
Number_of_associate_professors	0
Number_of_assistant_professors	0
Number_of_instructors	0
Number_of_faculty_all_ranks	0
FICE	0
College_name	0
State	0

Auto MPG Dataset

Missing Values

Column	Missing Values
uri	0
cylinders	0
displacement	0
horsepower	0
weight	0
acceleration	0
car	0
label	0
label_num	0

Top Attributes Most Correlated with 'label_num'

Attribute	Correlation
acceleration	0.491
cylinders	-0.790
displacement	-0.808
horsepower	-0.822
weight	-0.845

Cars93 Dataset

Missing Values

Column	Missing Values
car	0
Type	0
Minimum_Price	0
Midrange_Price	0
City_MPG	0
Highway_MPG	0
Air_Bags	0

Top Attributes Most Correlated with 'label_num'

Attribute	Correlation
City_MPG	0.735
Highway_MPG	0.721
Midrange_Price	-0.634
Minimum_Price	-0.620
Weight	-0.537

A.2 OWL Graph Queries

A.2.1 Full Graph Query

```
CONSTRUCT { ?s ?p ?o . }
WHERE { VALUES ?s { uris_str }
        ?s ?p ?o .
        FILTER (!regex(str(?p), "wiki", "i") &&
                !regex(str(?p), "pic", "i") &&
                !regex(str(?p), "thumb", "i") &&
                !regex(str(?p), "alt", "i"))
}
```

A.2.2 Simple Graph Query

```
CONSTRUCT {  ?s ?p ?o . }
WHERE { VALUES ?s { uris_str }
        ?s ?p ?o .
        FILTER(?p = rdf:type)
}
```

A.3 OWL2Vec* Extended Code

```
# Adding Explicit Numeric Literals
properties = self.onto.queryGraph(self.getQueryForUniquePropertiesWithNumericObject())

for prop in properties:
    self.__addTriple__(prop[0], RDFS.range, XSD.string)
    # logging.info('Property: ' + str(prop[0]))
    results = self.onto.queryGraph(self.getQueryForTriplesWithSpecificProperty(
                        property_uri=prop[0]))
    df = pd.DataFrame(data=results, columns=['URI', 'Property', 'Value'])
    df['Property'] = [prop[0]]*len(df)
    try:
        df['Value'] = pd.to_numeric(df['Value'], errors='coerce')
        # Calculate quantiles
        q = df['Value'].quantile([0, 0.4, 0.6, 0.9, 1])
        df.loc[df['Value'] < q.values[1], 'Value_cat'] = 'low'
        df.loc[(df['Value'] >= q.values[1]) &
                df['Value'] < q.values[2], 'Value_cat'] = 'medium'
        df.loc[(df['Value'] >= q.values[2]) &
                df['Value'] < q.values[3], 'Value_cat'] = 'high'
        df.loc[(df['Value'] >= q.values[3]), 'Value_cat'] = 'very high'

        for _, row in df.iterrows():
            try:
                self.__addTriple__(URIRef(row['URI']), URIRef(row['Property']),
                Literal(row['Value_cat']))
            except AttributeError:
                pass
    except Exception:
        logging.info('Could not add explicit numeric property')
```

```
# Adding Non-numeric and Implicit Numeric Properties
properties = self.onto.queryGraph(self.getQueryForUniquePropertiesWithNonNumericObject())

for prop in properties:
    self.__addTriple__(prop[0], RDFS.range, XSD.string)
    logging.info('Property: ' + str(prop[0]))
    results = self.onto.queryGraph(self.getQueryForTriplesWithSpecificProperty(
                        property_uri=prop[0]))
    entries = []

    # String or boolean properties
    for row in results:
        try:
            obj_value = float(row[2])
            logging.info('IT WORKED')
            entry = {'URI': row[0], 'Property': prop[0], 'Value': obj_value}
            entries.append(entry)
        except:
            logging.info('In first except')
            try:
                self.__addTriple__(row[0], prop[0], row[2])
            except AttributeError:
                pass

    # logging.info(results)
    # Numeric properties encoded as strings
    df = pd.DataFrame.from_dict(entries)
    try:
        df['Value'] = pd.to_numeric(df['Value'], errors='coerce')
        # Calculate quantiles
        q = df['Value'].quantile([0, 0.4, 0.6, 0.9, 1])
        df.loc[df['Value'] < q.values[1], 'Value_cat'] = 'low'
        df.loc[(df['Value'] >= q.values[1]) &
                df['Value'] < q.values[2], 'Value_cat'] = 'medium'
        df.loc[(df['Value'] >= q.values[2]) &
                df['Value'] < q.values[3], 'Value_cat'] = 'high'
        df.loc[(df['Value'] >= q.values[3]), 'Value_cat'] = 'very high'

        for _, row in df.iterrows():
            try:
                self.__addTriple__(URIRef(row['URI']), URIRef(row['Property']),
                Literal(row['Value_cat']))
                logging.info('Adding: ' + row)
            except AttributeError:
                pass
    except Exception:
        logging.info('')
        logging.info('Numeric Literal Not added')
        logging.info(prop)
```

A.4 Experiment Results

A.4.1 Classification Results

Dataset	Query Type	Mods	F1 Score	Baseline
Forbes	Simple	None	0.78	0.79
		Property High Corr	0.85	
		Attribute High Corr	0.83	
		Class Low Corr	0.81	
		Class Moderate Corr	0.82	
		Class High Corr	0.93	
	Full	None	0.78	
		Property High Corr	0.84	
		Class Low Corr	0.82	
		Class Moderate Corr	0.81	
		Class High Corr	0.92	
		Less Props	0.77	

Table 11: OWL2Vec* Classification Results for Forbes

Dataset	Query Type	Mods	F1 Score	Baseline
Auto 93	Simple	None	0.82	0.79
		Property High Corr	0.73	
		Attribute High Corr	0.71	
		Class Low Corr	0.38	
		Class Moderate Corr	0.58	
		Class High Corr	0.57	
	Full	None	0.76	
		Property High Corr	0.52	
		Class Low Corr	0.40	
		Class Moderate Corr	0.49	
		Class High Corr	0.52	
		Less Props	–	

Table 12: OWL2Vec* Classification Results for Auto 93

Dataset	Query Type	Mods	F1 Score	Baseline
Auto MPG	Simple	None	0.62	0.76
		Property High Corr	0.64	
		Attribute High Corr	0.70	
		Class Low Corr	0.76	
		Class Moderate Corr	0.84	
		Class High Corr	0.90	
	Full	None	0.81	
		Property High Corr	0.75	
		Class Low Corr	0.76	
		Class Moderate Corr	0.81	
		Class High Corr	0.89	
		Less Props	–	

Table 13: OWL2Vec* Classification Results for Auto MPG

Dataset	Query Type	Mods	F1 Score	Baseline
AAUP	Simple	None	0.80	0.80
		Property High Corr	0.76	
		Attribute High Corr	0.82	
		Class Low Corr	0.79	
		Class Moderate Corr	0.80	
		Class High Corr	0.89	
	Full	None	0.78	
		Property High Corr	0.80	
		Class Low Corr	0.79	
		Class Moderate Corr	0.74	
		Class High Corr	0.88	
		Less Props	0.80	

Table 14: OWL2Vec* Classification Results for AAUP

Dataset	Query Type	Mods	F1 Score	Baseline
Forbes	Simple	None	0.79	0.79
		Property Low Corr	0.80	
		Property Moderate Corr	0.80	
		Property High Corr	0.89	
		Class High Corr	0.77	
		Attribute High Corr	0.75	

Table 15: TransE Classification Results for Forbes

Dataset	Query Type	Mods	F1 Score	Baseline
Auto 93	Simple	None	0.69	0.79
		Property Low Corr	0.46	
		Property Moderate Corr	0.52	
		Property High Corr	0.81	
		Class High Corr	0.68	
		Attribute High Corr	0.67	

Table 16: TransE Classification Results for Auto 93

Dataset	Query Type	Mods	F1 Score	Baseline
Auto MPG	Simple	None	0.67	0.76
		Property Low Corr	0.75	
		Property Moderate Corr	0.76	
		Property High Corr	0.97	
		Class High Corr	0.80	
		Attribute High Corr	0.74	

Table 17: TransE Classification Results for Auto MPG

Dataset	Query Type	Mods	F1 Score	Baseline
AAUP	Simple	None	0.76	0.80
		Property Low Corr	0.81	
		Property Moderate Corr	0.71	
		Property High Corr	0.91	
		Class High Corr	0.78	
		Attribute High Corr	0.79	

Table 18: TransE* Classification Results for AAUP

A.5 Regression Results

Dataset	Query Type	Mod	RMSE	MAE	Baseline RMSE	Baseline MAE
Forbes	Simple	None	63.71	45.71	57.23	45.70
		Property High Corr	61.55	45.44		
		Attribute High Corr	60.36	44.79		
		Class Low Corr	60.80	45.87		
		Class Moderate Corr	62.42	47.32		
		Class High Corr	59.47	42.51		
	Full	None	62.19	46.33		
		Property High Corr	63.70	45.47		
		Attribute High Corr	60.36	44.79		
		Class Low Corr	62.09	46.21		
		Class Moderate Corr	60.84	44.52		
		Class High Corr	58.21	41.87		
		Less Props	62.42	45.91		

Table 19: OWL2Vec* Regression Results for Forbes

Dataset	Query Type	Mod	RMSE	MAE	Baseline RMSE	Baseline MAE
Auto 93	Simple	None	10.08	7.93	29.68	24.84
		Property High Corr	10.81	8.73		
		Attribute High Corr	8.29	6.33		
		Class Low Corr	9.50	7.89		
		Class Moderate Corr	9.93	7.96		
		Class High Corr	12.28	10.85		
	Full	None	12.06	9.23		
		Property High Corr	9.77	7.47		
		Class Low Corr	10.54	8.50		
		Class Moderate Corr	11.40	10.01		
		Class High Corr	9.51	7.92		

Table 20: OWL2Vec* Regression Results for Auto 93

Dataset	Query Type	Mod	RMSE	MAE	Baseline RMSE	Baseline MAE
Auto MPG	Simple	None	13.21	10.59	34.79	28.47
		Property High Corr	14.04	10.80		
		Attribute High Corr	13.30	9.93		
		Class Low Corr	13.96	11.60		
		Class Moderate Corr	14.00	10.74		
		Class High Corr	10.44	8.23		
	Full	None	15.87	12.14		
		Property High Corr	15.53	11.66		
		Class Low Corr	14.31	10.63		
		Class Moderate Corr	12.98	10.15		
		Class High Corr	12.93	9.96		

Table 21: OWL2Vec* Regression Results for Auto MPG

Dataset	Query Type	Mod	RMSE	MAE	Baseline RMSE	Baseline MAE
Forbes	Simple	None	75.26	55.90	57.23	45.70
		Property Low Corr	83.03	59.73		
		Property Moderate Corr	75.41	54.26		
		Property High Corr	61.01	44.60		
		Attribute High Corr	70.23	52.40		
		Class High Corr	70.11	51.30		

Table 22: TransE Regression Results for Forbes

Dataset	Query Type	Mod	RMSE	MAE	Baseline RMSE	Baseline MAE
Auto 93	Simple	None	12.29	9.55	29.68	24.84
		Property Low Corr	13.11	11.29		
		Property Moderate Corr	14.18	11.61		
		Property High Corr	10.99	9.27		
		Attribute High Corr	11.55	8.66		
		Class High Corr	12.53	10.42		

Table 23: TransE Regression Results for Auto 93

Dataset	Query Type	Mod	RMSE	MAE	Baseline RMSE	Baseline MAE
Auto MPG	Simple	None	13.55	9.97	34.79	28.47
		Property Low Corr	11.07	7.57		
		Property Moderate Corr	13.58	9.38		
		Property High Corr	11.08	9.06		
		Attribute High Corr	9.33	7.03		
		Class High Corr	13.30	10.54		

Table 24: TransE Regression Results for Auto MPG