Algorithmic Taxonomical Hierarchy Using Hypernymy Detection

A B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Manan Gupta
(170101035)

under the guidance of

Dr. Ashish Anand

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Algorithmic Taxonomical Hierarchy Using Hypernymy Detection**" is a bonafide work of **Manan Gupta** (**Roll No. 170101035**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Ashish Anand**

Associate Professor,

Department of Computer Science & Engineering,

Indian Institute of Technology Guwahati, Assam.

May, 2021

Guwahati.

Acknowledgements

I would like to thank my supervisor, Dr. Ashish Anand for his support during the course of the BTP. I would also like to thank Mr. Akshay Parekh for his invaluable inputs and guidance.

Contents

List of Figures

List of Tables

Abstract

Relation Extraction (RE) is the task of extracting semantic relationships from text, which usually occur between two or more entities. These relations can be of different types. E.g. *“Delhi is the capital of India”* states a *“is the capital of”* relationship from Delhi to India. This can be denoted using triples, *(Delhi, is the capital of, India).*

Among these relations too there exists relationships. These relationships allow us to arrange the relations in a hierarchy. In this report, we focus on is-a relationships. Such a hierarchy of is-a relationships (also known as hypernymy relationships) as an acyclic directed graph is extremely important in tasks like information extraction and question answering. This will allow us to infer more general relations more already provided information. For example if already know that *”Delhi is the capital of India”*, then we should be able to answer the question *”Is Delhi in India”* correctly. This would be possible only if our system was aware that being a capital of a country automatically implies that it is part of the country. So if we add a hypernymy edge from *is the capital of* to *is in* then we would be able to answer such queries.

There have been very few studies dealing with this task of creating a hierarchy of relations. One such algorithm is *PATTY* but it has some shortcomings that we will discuss in this report. We then run *PATTY* on *DBpedia* knowledge base and try to improve its performance by making modifications to the algorithm. We also try to model the data as a graph and use machine learning techniques like *TransE* to create an alternate algorithm for the task of creating a hierarchy algorithmically from a set of relations.

We conclude the report with comparisons of the various algorithms explored in the work and some future works possible.

Chapter 1

Introduction

Relations between two or more entities also have relationships among them. As discussed in the abstract, the relationships that we are going to focus on are is-a relationships. These are the relationships where having one relation guarantees that the parent relation would also hold between the entities in question. Figure 1.1 shows a small snippet of a hierarchy consisting of relations between two people.

This task of organizing binary relations as a taxonomy is central to many tasks like information extraction and question answering. This is described in further detail in the works of [PAA20]. The hierarchy allows us to infer more information based on the ones that we already have. For example, consider the relation *X is a sister of Y*. If we have a hierarchy as shown in Figure 1.1, we can immediately infer that X is also the sibling of Y. They are relatives and therefore also know each other. We can now use this inferred knowledge to answer a wider range of questions like *Is Y a relative of X?*, *Do Y and X know each other?* and *Are Y and X siblings?* among others.

There are only a few studies that deal with this task. There also exist semantic taxonomies like WordNet [Fel98], but they are heavily limited in terms of their domain and scope. Therefore, it is the need of the hour to have automated methods to create the

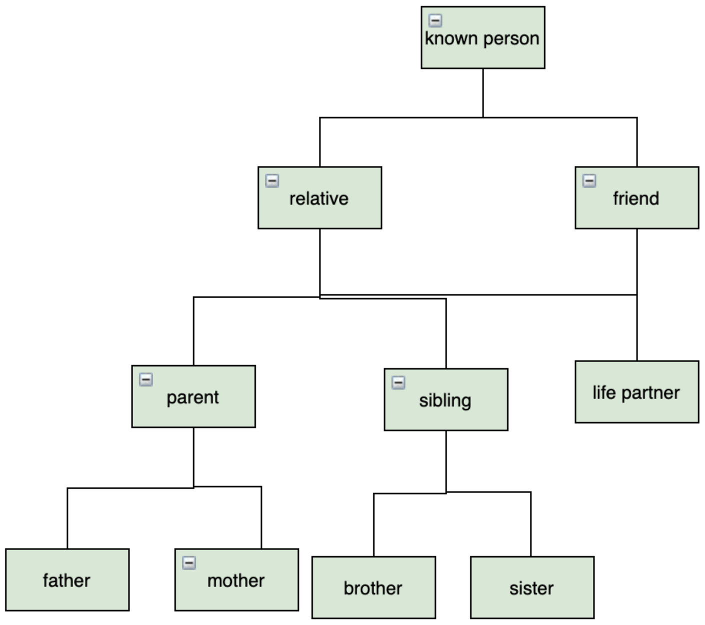

Fig. 1.1 Hierarchy of relations

hierarchy for us given the relations as inputs. One such algorithm is *PATTY* as proposed in [NWS12]. Data sources like Wikipedia Infoboxes [1], DBpedia [LIJ$^+$14], and Wikidata [VK14] have multiple relations that span almost every genre and topic and have wide coverage. In this report, we use DBpedia [LIJ$^+$14] as the primary source and try to improve upon the existing algorithm for taxonomy creation as proposed in [NWS12]. We also explore the field of representing the entities and relations as a graph and then using machine learning techniques described in [BUGD$^+$13] to find the embeddings that we use in the creation of the taxonomy.

[1] `https://en.wikipedia.org/wiki/Wikipedia:List_of_infoboxes`

1.1 Problem Statement

Extract the binary relation triplets from DBpedia [LIJ$^+$14]. Propose an algorithm that uses them as input to create a hierarchy of the binary relations consisting of is-a relations.

1.2 Contributions

We start by extracting all the triplets from DBpedia and storing them in MySQL. We run the existing *PATTY* algorithm on this dataset. We then try to improve the algorithm by various means that we will discuss in this report. We also try a novel approach of viewing the dataset as a graph and using machine learning techniques like *TransE*. Finally, we compare all the different approaches and conclude with future works.

1.3 Organization of The Report

This chapter introduces the problem statement covered in this report. We described the taxonomy creation of binary relations and its applications. The rest of the chapters are organised as follows: next chapter we provide a review of prior works. In Chapter 3, we describe the dataset and setup the benchmark to compare our results against. In Chapters 4 and 5, we discuss various improvements to PATTY and the application of TransE to our task respectively. And finally, in chapter 6, we conclude with some future works.

Chapter 2

Review of Prior Works

To handle the task described, all the pertaining research papers and their methods have been summarized here. These provide an overview of the work done in the field and then we describe our contributions in the subsequent chapters.

2.1 PATTY - A Taxonomy of Relational Patterns with Semantic Types

This subsection presents the works of [NWS12]

2.1.1 Overview

Most of the work done in the field is restricted to relations that can be specified with single words and do not contain entire patterns or phrases. Their work deals with extracting relational phrases and patterns from large corpora and create a taxonomy out of them. For example, the pattern *X is romantically involved with Y* is synonymous with the pattern *X is dating Y* and both are subsumed by *X knows Y*. To accomplish their objective, they extract SOL patterns from the corpora. SOL patterns are an expressive family of relational patterns, which combine syntactic features (S), ontological type signatures (O), and lexical features (L). After extracting the SOL patterns, they create a hierarchy of them by defining soft set inclusion on the support sets of the patterns.

2.1.2 Method

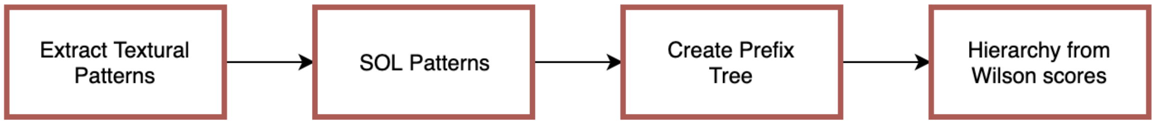

Fig. 2.1 Various Steps in PATTY Hierarchy Creation [NWS12]

The entire process is shown in Figure 2.1.

The first step is to extract **textual patterns**. They use the Stanford Parser on individual sentences to obtain dependency paths. They then detect named entities in the parsed corpus and find the shortest path between two named entities detected in a sentence. They further augment the path with adverbial and adjective modifiers.

From these textual patterns, we get **SOL patterns** by adding to the sequence of words, POS tags, wildcards, and ontological types. An ontological type is a semantic class name (such as *singer*) that stands for an instance of that class. The support set of a pattern is described as the pairs of entities that appear in place of the entity placeholders in the corpus.

There are two types of pattern generalization that the authors describe.

- **Syntactic Generalization:** achieved by replacing words with POS tags, introducing wildcards, and generalizing the terms in the pattern.

- **Semantic Generalization:** meaning of one pattern is more general than the meaning of the others. For example - *is the capital of* is a more general pattern than *is a city in.*

Further, they define soft set inclusion between different sets using the Wilson score, which is used in place of the naive estimate $|S \cap B|/|S|$, because the naive estimate is susceptible to overestimating the inclusion score in case of support set size.

Finally, they create a prefix tree of all the patterns using their support sets in the decreasing order of their occurrence frequency. The expectation is that if pattern A is a hyponym of pattern B, then the support set of B would contain a lot of elements from the prefix of the support set A. Finally, the soft set inclusion scores are used to create a hierarchy of the patterns. For eliminating redundancy and preserving the acyclic structure of the graph they use a greedy algorithm wherein they only add an edge to the graph if follows the two invariants.

2.1.3 Results

The PATTY taxonomy comprises 350,569 pattern synsets with a random sampling-based evaluation showing a pattern accuracy of 84.7%. PATTY has 8,162 subsumptions, with a random sampling-based precision of 75%.

2.1.4 Strengths

- The entire process is fully automated and the user is only required to change the hyper-parameters.
- First project that worked on pattern recognition and hierarchy creation that created such a large hierarchy.

2.1.5 Weaknesses

- Only uses the support sets to decide on hierarchy creation.
- Cascading errors - Errors in relation extraction are also seen in taxonomy creation

2.2 Taxonomical hierarchy of canonicalized relations from multiple Knowledge Bases

This subsection presents the works of [PAA20].

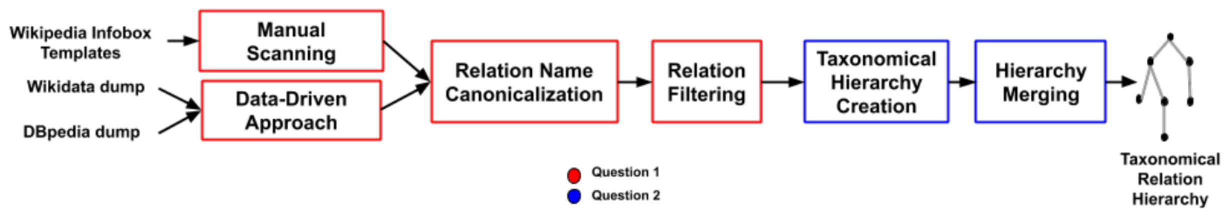
Wikipedia Infobox Templates
Wikidata dump
DBpedia dump
Manual Scanning
Data-Driven Approach
Relation Name Canonicalization
Relation Filtering
Taxonomical Hierarchy Creation
Hierarchy Merging
Taxonomical Relation Hierarchy
Question 1
Question 2

Fig. 2.2 Various Steps in Manual Hierarchy Creation [PAA20]

2.2.1 Overview

This work focuses on the creation of an unambiguous taxonomy for relations using three different sources *Wikipedia Infobox*, *DBpedia*, *Wikidata*. The study presented focuses on relations between 3 different entity types *person*, *location*, and *organization*.

2.2.2 Method

The overall model is illustrated in Figure 2.2. The first step is to extract relations from the sources. This is done manually for Wikipedia Infobox and in a data-driven way from DBpedia and Wikidata. The entity types for these relations are restricted to *person*, *location*, and *organization*.

This is followed by the manual canonicalization of the relation names. This is required because many relations have different representations in different resources. For example, the relation *place of burial* takes 3 different forms, *burial_place* in Wikipedia Infobox, *place of burial* in DBpedia, and *placeOfBurial* in Wikidata.

After this step, the authors create a hierarchy for each of the 3 sources. This is done manually by the authors based upon their collective judgment. The hierarchy they created had a maximum depth of 5. The first 3 layers are -

- Depth 0: root node.
- Depth 1: Head entity type of a relation.
- Depth 2: Tail entity type of a relation.

This creates a total of nine buckets since there are three types for both head and tail. All the relations are distributed across them.

The final step is to merge the hierarchies from all three sources into one. The authors also provide a detailed analysis of the coverage of various resources and distribution of relations across these various resources.

2.2.3 Results

The authors are successful in creating a hierarchy of 623 relations. They observe that their hierarchy subsumes 85% of relations from relation extraction datasets with restricted entity types.

2.2.4 Strengths

- Since the hierarchy is manually created, there is as little error as possible.
- Can be used by other works as a gold standard to evaluate their work
- Can also be used for training a supervised algorithm.

2.2.5 Weaknesses

- The process is not automated and will not scale to hierarchy creation without restricting the entity types.

2.3 Translating Embeddings for Modeling Multi-relational Data

This subsection presents the works of [BUGD$^+$13].

Their work deals with embedding entities and relationships in a low-dimensional vector space. They propose **TransE**, a method that models relationships by interpreting them as translations operating on the low-dimensional embeddings of the entities.

The input for their method is a list of triplets of the form *(head, label, tail)* where *head* and *tail* are entities and the *label* is a relation. They interpret this data as a directed graph where there is an edge from *head* to *tail* with the *label*. The learned embeddings can be used for other natural language processing tasks like the addition of new facts to existing knowledge bases without requiring extra knowledge.

Their proposed algorithm is fast enough to work with a large-scale data set with 1M entities, 25K relationships, and more than 17M training samples.

2.4 Conclusion

This chapter provided details of the existing algorithm **PATTY** that we will try to improve. Further, it summarized the works of [PAA20] in the creation of a taxonomy that we will be using as a gold standard. Finally, this section concludes by introducing a method to find embeddings of the relations and the entities which we will employ in our efforts to improve **PATTY**.

Chapter 3

Base Benchmark

In this section, we describe the parsing of DBpedia [LIJ$^+$14] and running it against **PATTY** which will serve as our benchmark. We then describe all the improvements that we try for improving the algorithm from the next chapter onwards.

3.1 Dataset and Preprocessing

In this section, we describe the process of using the DBpedia dataset and preprocessing required to work with it. The first step is to download the DBpedia as tables from `https://wiki.dbpedia.org/DBpediaAsTables#h347-2`. There are a total of **463** files in the downloaded folder. All these files are CSV files that are used to extract all the entities, relation triplets which are stored in MySQL.

3.1.1 MySQL Schemas

We store all the relationship triplets in a table called *dbpedia_relations*. The schema of the table is given in Table 3.1. *lhs* and *rhs* are the left-hand side and right-hand side entities respectively in the triplet. *binaryRelation* is the binary relation between the two entities.

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
id	int(11)	NO	PRI	NULL	auto_increment
lhs	varchar(300)	YES		NULL	
binaryRelation	varchar(300)	YES	MUL	NULL	
rhs	varchar(300)	YES		NULL	

Table 3.1: Schema of *dbpedia_relations*

Few examples of the data stored in *dbpedia_relations* are showed in Table 3.2.

id	**lhs**	**binaryRelation**	**rhs**
21232792	Gene Paulette	position	Infielder
17246131	AFL Ontario	country	Canada
94514733	Kwoun Sun-tae	height	1.83
101811538	Ford River Township Michigan	areaWater	1200000.0
41406888	Waccamaw silverside	phylum	Chordate

Table 3.2: Examples from *dbpedia_relations*

Further, we store the type list for all entities that dbpedia offers in another table called *dbpedia_types*. The schema of the table is given in Table 3.3. We also add indexes on *binaryRelation* in *dbpedia_relations* and *value* in *dbpedia_types* for faster accesses.

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
id	int(11)	NO	PRI	NULL	auto_increment
value	varchar(300)	YES	MUL	NULL	
typeList	varchar(300)	YES		NULL	

Table 3.3: Schema of *dbpedia_types*

As mentioned in section 2.2, [PAA20] restrict the entity types to person, location and organization. We do the same and store the restricted entities in the table *dbpedia_restricted_entities*. The schema of the table is given in Table 3.4.

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
value	varchar(300)	YES	MUL	NULL	

Table 3.4: Schema of *dbpedia_restricted_entities*

Few examples of the data stored in *dbpedia_restricted_entities* are shown in Table 3.5.

value
Derrick Robinson
Patrick J. Que Smith
Hohenau
Johnny Johnston (cricketer)
Jamie Howarth
Joseph Cortese
German People's Party (1868)
Stuart Wilkinson (rugby league)
Borjeleh
Gao Huaide

Table 3.5: Examples from *dbpedia_restricted_entities*

PATTY algorithm uses the size of support sets and the intersection of the support sets in the construction of the hierarchy. Therefore, we store the size of the support sets for the relations with restricted entities in a MySQL table called *dbpedia_support_size_restricted*. The schema of the table is given in Table 3.6. We also store the size of intersections of the support sets in *dbpedia_intersect_size_restricted*.

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
binaryRelation	varchar(300)	NO	PRI	NULL	
size	int(11)	YES		NULL	

Table 3.6: Schema of *dbpedia_support_size_restricted*

Few examples of the data stored in *dbpedia_support_size_restricted* are shown in Table 3.7.

binaryRelation	**size**
successor	254345
hipSize	0
assistantPrincipal	20
date	0
analogChannel	8

Table 3.7: Examples from *dbpedia_support_size_restricted*

The schema of the table *dbpedia_intersect_size_restricted* is given in Table 3.8.

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
binaryRel1	varchar(300)	NO	PRI	NULL	
binaryRel2	varchar(300)	NO	PRI	NULL	
size	int(11)	YES		NULL	

Table 3.8: Schema of *dbpedia_intersect_size_restricted*

Few examples of the data stored in *dbpedia_intersect_size_restricted* are shown in Table 3.9.

binaryRel1	**binaryRel2**	**size**
routeEnd	routeJunction	906
architect	owner	5
leftTributary	sourcePlace	36
sourceMountain	state	6
party	successor	3

Table 3.9: Examples from *dbpedia_intersect_size_restricted*

3.1.2 Dataset Description

We will use the triplets from DBpedia with restricted entity types for all our experiments since we have the gold standard available for them. We will be using the F1 score as the primary value to maximize. The description of the dataset is given in Table 3.10.

Number of entities	22,69,358
Number of relations	551
Number of triplets	2,44,98,082

Table 3.10: Dataset Description

There are a total of **65** is-a relations in the true hierarchy constructed by [PAA20].

3.2 PATTY Results

We ran the PATTY algorithm on the dataset by using the size of support sets and the size of the intersection of the support sets to find the Wilson scores. We did not need to use the prefix tree since we could afford to computationally find all the intersections. We keep the cutoff above which we will accept the relation pair as a hierarchy link as a variable and find its optimal value while maximizing the F1 score. The results of running PATTY are shown in Table 3.11.

Best F1 Score	0.311926
Cutoff	0.09504
Number of relations in Hierarchy	44
Precision	0.38636
Recall	0.26153

Table 3.11: PATTY Results

3.2.1 Examples that fail in PATTY

In this section, we look at a few class of examples that fail in PATTY and try to reason about why this happened.

1. The first classes of examples is of those that have very similar support sets but are not related by an is-a relation.

   We look at *iataLocationIdentifier* and *faaLocationIdentifier* in detail.

   According to Wikipedia [1], IATA location identifier is a three-letter geocode used to designate many airports and metropolitan areas around the world. It is defined by the International Air Transport Association.

   Similarly, Wikipedia [2] defines FAA location identifier as a 3 to 5 character alphanumeric code identifying aviation-related facilities inside the United States.

   There are a lot of airports that have the same FAA and IATA location identifier as seen in Table 3.12. This leads to a high size of the intersection of the support sets of both the relations and since PATTY only relies on the size of intersections, it mistakes them to be in an is-a relation.

[1] https://en.wikipedia.org/wiki/IATA_airport_code
[2] https://en.wikipedia.org/wiki/Location_identifier#FAA_identifier

Name	**Code**
Florence Regional Airport	FLO
Gwinnett County Airport	LZU
McAlester Regional Airport	MLC
Phoenix Deer Valley Airport	DVT
Skwentna Airport	SKW

Table 3.12: Some airports with the same FAA and IATA location identifiers

lieutenancyArea and *councilArea* are two other relations that are misjudged by PATTY for the same reason.

2. Next we take a look at examples that fail because their support sets are too small. *nationalTeam* and *team* are one example. In the hierarchy created by [PAA20] they form an is-a relation but PATTY is unable to find this because the support set of *nationalTeam* is only 4 which is very small.

We now move onto improving the results of PATTY by changing the algorithm.

Chapter 4

Improving PATTY

4.1 Addition of Meaning of Relations

Looking at the second class of examples that fail in PATTY in the previous chapter, it stands to reason that we require more than just the support sets to make correct predictions in the creation of the hierarchy. Motivated by this thought, we set out to also incorporate the meaning of relation name in the algorithm. This we accomplish in 3 steps which are now explained.

4.1.1 Steps

Step - 1. Splitting Words

The relation names in DBpedia [LIJ$^+$14] are stored in camel case. For example, *winsAtOtherTournaments* and *unitedStatesNationalBridgeId*. We split the relation name at capital letters and get the words.

$$\textit{winsAtOtherTournaments} \implies \text{wins at other tournaments}$$
$$\textit{unitedStatesNationalBridgeId} \implies \text{united states national bridge id}$$

Step - 2. Relation Embeddings

After splitting the relation names into words we use the GloVe embeddings created by [PSM14]. The embeddings are downloaded from
`https://nlp.stanford.edu/projects/glove/`.
We get the embeddings for the relations by taking the average of all the word embeddings of the constituent words in the relation name as seen in equation 4.1.

$$emb(rel) = \frac{\sum_{\forall x \, \epsilon \, words(rel)} emb(x)}{len(words(rel))} \tag{4.1}$$

Step - 3. Additional Signal

Finally, we use the relation embeddings to find the cosine similarity between the two embeddings as use this value as an additional signal. We weight both of these signals, one from PATTY and the other from this similarity score. The weight provided to PATTY is α which is kept as a variable. We optimize this along with the cutoff to find the maximum F1 score.

4.1.2 Results & Comparison

The results of this experiment are summarized in Table 4.1.

Alpha	0.95
Best F1 Score	0.315789
Cutoff	0.2585
Number of relations in Hierarchy	30
Precision	0.5
Recall	0.230769

Table 4.1: PATTY With Relational Embeddings Results

The comparison of this algorithm's results with PATTY can be seen in Figure 4.1.

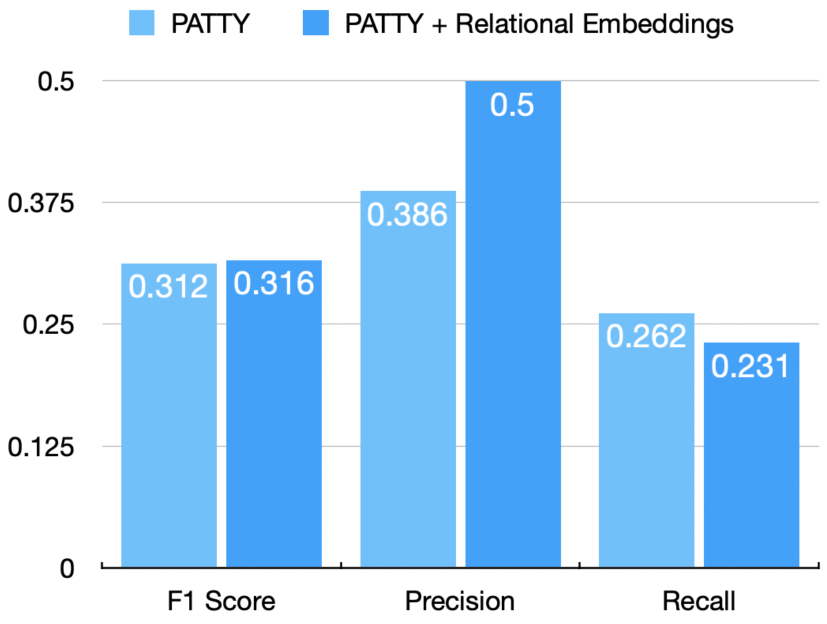

Fig. 4.1 Comparison of PATTY & PATTY+Relational Embeddings

We can see from figure 4.1 that the F1 score of our approach is slightly better than PATTY. The precision of our approach has indeed increased as we had hypothesized but the recall has decreased in the process.

4.2 Weighted Entities

In this section, we try out an alternate experiment to improve the results of PATTY. This experiment was motivated by the first class of errors. We try to solve them by weighting the entities in such a way that more common entities get assigned lower weight.
PATTY uses the intersection of support sets and the size of support sets to make decisions for hierarchy creation. One possible issue with PATTY is that it assigns equal weight to all the entities. This restriction can be relaxed though by adding an appropriate weighting schema. This is exactly what we will try in this section.

4.2.1 Weighting Scheme

We provide weights to all entities based on the number of relation support sets it is part of. The idea behind such a weighting scheme is that if an entity is part of fewer support sets then it must be providing more specific domain information than other entities part of more support sets. We also add another restriction to our weighting scheme that all the weights should be less than 1.
With all these constraints and intuition, the weighting scheme becomes apparent which are specified in equations 4.2.

$$\begin{aligned} count(ent) &= \sum_{\forall rel} \begin{cases} 1, & \text{if } ent \ \epsilon \ support\_set(rel) \\ 0, & \text{otherwise} \end{cases} \\ weight(ent) &= \frac{total\_relations - count(ent)}{total\_relations} \end{aligned} \tag{4.2}$$

With these weighted entities, we can also define the weight of pair of entities which form the support set of a relation specified in 4.3.

$$weight(ent1, ent2) = weight(ent1) * weight(ent2) \tag{4.3}$$

4.2.2 Results

We now use these weighted pairs in the support set to find the weighted Wilson scores with the rest of the algorithm remaining the same. The results are shown in Table 4.2.

Best F1 Score	0.311926
Cutoff	0.09504
Number of relations in Hierarchy	44
Precision	0.38636
Recall	0.26153

Table 4.2: PATTY With Weighted Entities Results

4.2.3 Analysis

The results are the same as PATTY! It turned out that there were very few entities that occurred in the support sets of a lot of relations. Most of the entities had a score very close to 1 which led to a very minor difference in the weighted Wilson score with respect to the original PATTY algorithm.

Chapter 5

Neural Network-Based Approaches using TransE

In this section, we explore the application of TransE introduced in section 2.3 to our task. The implementation of TransE is available online at `http://openke.thunlp.org/`.

5.1 Similarity-Based Approach

Running the TransE algorithm provides us with embeddings for entities and the relations of size **200** each. We use the embeddings of the relations and find the similarity scores between them. We use this as the signal for creating the hierarchy.

The results are shown in Table 5.1.

Best F1 Score	0.0007927
Cutoff	-0.1019
Number of relations in Hierarchy	2458
Precision	0.0004068
Recall	0.015384

Table 5.1: TransE Embeddings Similarity Results

The results are not very good, but this was expected and reiterates our belief that is-a hierarchy is an inherently different task than similarity. There might be some correlation between the two tasks but just using the similarity as a measure for predicting is-a hierarchy does not provide good results. We now move onto a machine-learning-based approach.

5.2 Machine Learning Approach

After the failure of the similarity-based approach, it becomes apparent that we must use the embeddings of the relations differently. Keeping this in mind, we try out a machine learning-based approach.

We start by sampling **20%** values from the true hierarchy randomly. Furthermore, we add an equal number of negative samples to our training set.

Next, we describe the model that we use for predicting whether the two relations are in an is-a hierarchy. The input of the model is the embeddings of both the relations concatenated together. The model is described in Table 5.2.

Model: "Sequential"		
Layer (type)	**Output Shape**	**Param #**
dense (Dense)	(None, 20)	8020
dense_1 (Dense)	(None, 10)	210
dense_2 (Dense)	(None, 1)	11
Total params:	8,241	
Trainable params:	8,241	
Non-trainable params:	0	

Table 5.2: Model Description

We now train the model for **150** epochs and then use this model for the prediction of is-a hierarchy links.

The results of this approach are listed in Table 5.3.

Best F1 Score	0.04166
Cutoff	0.99471
Number of relations in Hierarchy	31
Precision	0.06451
Recall	0.03076

Table 5.3: TransE Embeddings ML-Based Results

5.3 Comparison

In this section, we compare the results of both the approaches we used for the TransE model. The comparison can be seen in Figure 5.1.

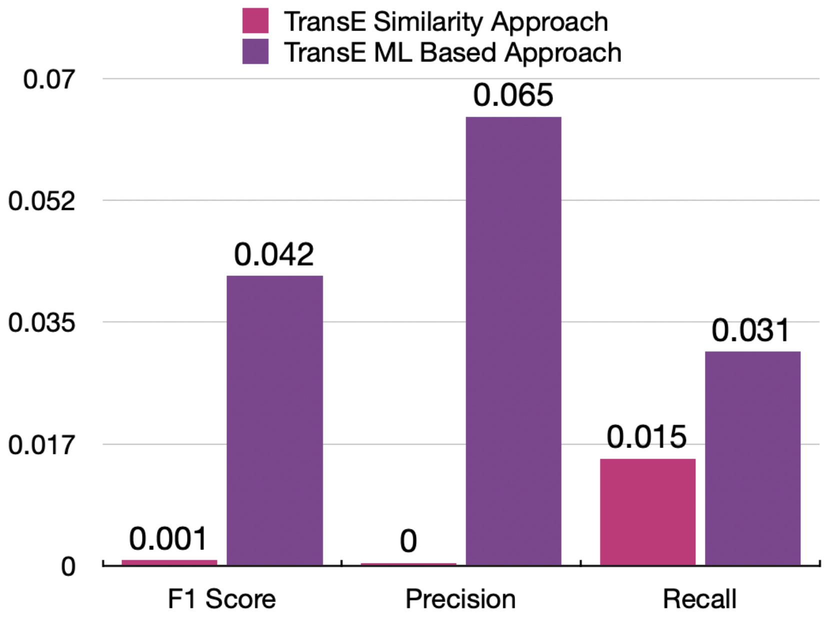

Fig. 5.1 Comparison of Similarity & ML Approaches

We can see from the figure that the ML-based approach significantly outperforms the similarity-based approach in all respects.

However, the results of the ML-based approach are nowhere nearly as good as that of PATTY. There are multiple reasons for this, the most significant being that the amount of data that we could train on was very small which led to a sub-par model.

In the next chapter, we discuss the conclusions and future works.

Chapter 6

Conclusion and Future Work

6.1 Results

In this section, we do a final comparison between all the different methods that we have discussed in this report. The comparison can be seen in Figure 6.1.

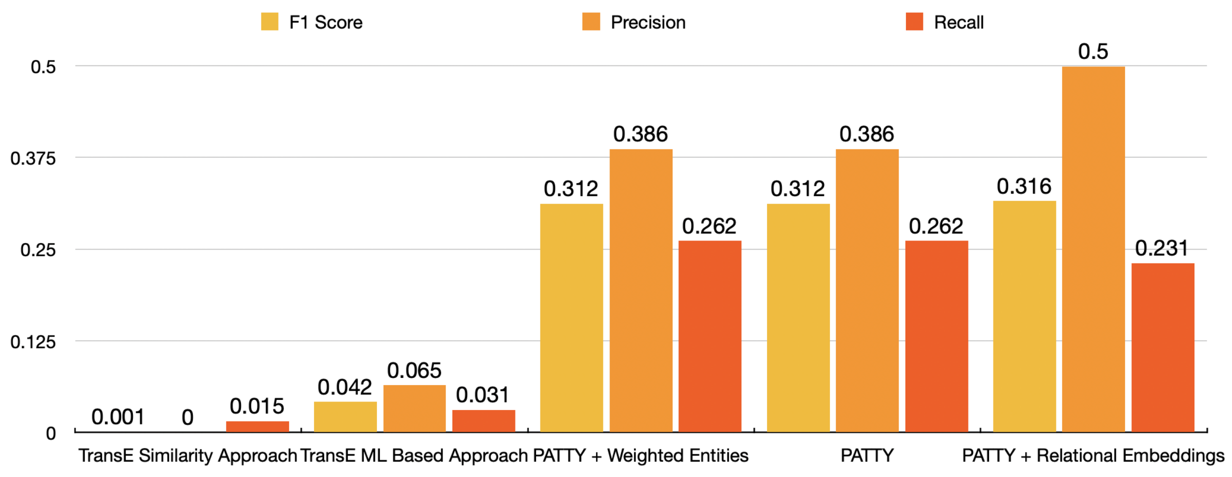

Fig. 6.1 Comparison of All Approaches

We can see that having relational embeddings with the original PATTY algorithm is the best algorithm but it is only slightly better than PATTY itself.

6.2 Future Work

The TransE approach applied to the task of hierarchy creation for binary relations seems to be a promising field for further exploration. The major problem due to which the results were poor was the lack of a large training dataset. This however can be mitigated by various methods like

- *Generating Synthetic Data:* We can try to generate more training samples from the existing ones using techniques like *Synthetic Minority Over-sampling Technique (SMOTE)* [CBHK02] and *Modified- SMOTE*.
- *Ensembling Techniques:* The idea is to aggregate multiple weak models. *Bagging* and *Boosting* are two techniques that can be used.

We could also try an integrated approach of TransE and PATTY using signals from both to try to get the best of both worlds.

References

[BUGD+13] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. 2013, 12 2013.

[CBHK02] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[Fel98] Christiane Fellbaum. Wordnet an electronic lexical database. *The MIT Press*, 1998.

[GW14] Adam Grycner and Gerhard Weikum. Harpy: Hypernyms and alignment of relational paraphrases. *COLING*, 2014.

[GWP+15] Adam Grycner, Gerhard Weikum, Jay Pujara, James Foulds, and Lisa Getoor. Relly: Inferring hypernym relationships between relational phrases. *EMNLP*, 2015.

[LIJ+14] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 6, 01 2014.

[NWS12] Ndapandula Nakashole, Gerhard Weikum, and Fabian Suchanek. Patty: A taxonomy of relational patterns with semantic types. *EMNLP*, 2012.

[PAA20] Akshay Parekh, Ashish Anand, and Amit Awekar. Taxonomical hierarchy of canonicalized relations from multiple knowledge bases. *CoDS-COMAD*, 2020.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[SGD16] Vered Shwartz, Yoav Goldberg, and Ido Dagan. Improving hypernymy detection with an integrated path-based and distributional method. *ACL*, 2016.

[VK14] Denny Vrandečić and Markus Krötzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, September 2014.