Utrecht University

Learning to Generate Ontologies

Using support vector machines to classify ontology triples

Romy A.N. van Drie

A thesis presented for the degree of
Master of Arts in Linguistics

Utrecht University
Languages, Literature and Communication
August 2020

External supervisor: Maaike de Boer, TNO
Internal supervisor: Tejaswini Deoskar, UU

Abstract

Ontology engineering is a strenuous task, which has fueled research investigating how an ontology can be generated from text. Ontologies store information in triples, which are also the targets of information extraction. This insight is used in this thesis to motivate using a method from information extraction to learn ontology triples. The transfer of methods is novel, as well as the scope of the triple extraction. Parts of the triple have been extracted with classifiers in previous studies, but in this thesis, a machine learning method is used to classify complete triples. Several classifiers are made to classify triples and arguments of triples. The features used are words, POS tags and dependency parses. The best-performing triple classifier achieves an F1 score of 0.520, outperforming the state of the art, albeit on an easier task. These results indicate that the novel approach taken in this thesis is promising.

Acknowledgements

This thesis would not have been possible without a number of people. First, the group of people that were most directly involved in this thesis; I want to thank Maaike for her endless support and good advice, Tejaswini for asking questions that stimulated critical thinking, and Roos for her fresh perspective and insights to professionalize my code.

I would like to thank the people I got to know at TNO, for taking an interest in my work, and perhaps even more importantly, for being so welcoming. I have learned something from each of you, and it has been a pleasure to write my thesis at TNO.

I owe a big thank you to Sjoerd and Rosita, for being there everyday, writing their theses alongside me. Writing a thesis during a pandemic would have been much harder without our morning zoom call routine.

Finally, I would like to express my gratitude to my friends and family for their support behind the scenes - not just while writing this thesis, but during my entire academic adventure; Pamela, for taking my mind off of things and being a true friend, my brother, for being someone to look up to, and my parents, for encouraging me to pursue my interests while preaching there is more to life than academic success.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis presents a new approach to learn information for an ontology. The approach is inspired by information extraction techniques. Ontologies and information extraction systems share a basic building block: the triple, which consists of two arguments and a relation between them. In this thesis, a classification approach is used to extract triples for an ontology. Classification has been used in the domain of information extraction to extract arguments, or relations, but not both. Thus, this thesis is innovative in two different ways. The first being that it carries over a method to a new domain. The second is that it extends previous classification approaches used in information extraction. The research question driving this thesis is how ontology triples can be classified using a machine learning classifier.

This thesis is organized in three main chapters: background, experimental study and discussion. Naturally, earlier chapters inform the choices made in later chapters and fuel the discussions there. However, the chapters are written such that they can be read in isolation if necessary.

In Chapter 2, the theoretical background is given that is needed to understand the motivation of experimental design choices and place the results and scope of the study in context. There are three main sections in this background chapter: ontologies, information extraction and classification.

In Chapter 3, the experimental study is laid out. An existing dataset is adapted and expanded with negative examples and more features so it can be used for the purpose of training classifiers. In this study, several different classifiers are trained on different features and tested to investigate how triples can be classified most adequately.

Chapter 4 wraps up this thesis. It contains a conclusion and discusses future research possibilities.

Chapter 2

Background

The work presented in this thesis is inspired by the need for ontologies and the high costs of creating them. As such, a theoretical background on ontologies is needed, which is given in the first section of this chapter. First, I explain what ontologies are, and highlight that the specific form of an ontology may vary from case to case. Highly expressive ontologies have different forms than less expressive ones. Then, the field of ontology learning is introduced. I argue that less expressive ontologies may be learned using techniques from information extraction (IE), given the structural similarities of an ontology triple and an IE triple. Information extraction is the topic of the second section of this chapter. It is explained what (open) information extraction is, why it can be compared to ontology learning, and the state of the art is described. In the third section of this chapter, I give background information on classification and highlight related work in which a classification approach is used for relation extraction and named entity recognition. I make the case that information extraction can be seen as a classification problem. While classification techniques have been used for relation extraction and named entity recognition, two subtasks of information extraction, they have not been used to extract IE triples.

2.1 Ontologies

An ontology is a formal model of the structure of a system, capturing relevant concepts of a specific domain and the relations between these concepts (Guarino, Oberle, & Staab, 2009). An example of such a system is a company, with different functions of employees as concepts, and the interpersonal relationships of people as relations. In this example, concepts of interest could be *manager* and *secretary*. A possible relation is that the secretary *assists* the manager. Specific employees are instances of a concept. To illustrate this, an example is given below. The concepts *manager* and *secretary* are a set containing specific individuals, indicated with a unique

number. The relation *assists* is a set with tuples that specify the two elements (people) engaging in that relation.

- manager = {85331, 10952, ...}
- secretary = {98090, 36561, ...}
- assists = {(98090, 85331), (36561, 10952), ...}

(example derived from Guarino et al. (2009))

The notion of an ontology was originally defined by Gruber (1995) as "an explicit specification of a conceptualization". A conceptualization is a simplified view of the world that one wants to represent. Every knowledge base, or knowledge-based system needs a conceptualization (Gruber, 1995). An ontology explicitly specifies the concepts and relations that are relevant for modelling a domain (Gruber, 2008). Borst (1997) defined an ontology as a "formal specification of a shared conceptualization". This definition requires that the conceptualization is expressed in a formal, machine readable format and that the conceptualization expresses a shared view. These requirements points to the application of ontologies. Ontologies are commonly used to exchange data among different systems, and enable interoperability between systems and databases. They also provide a means for query answering and accessing information in general (Gruber, 2008).

2.1.1 Expressivity of Ontologies

Ontologies can have many different forms and vary in their amount of expressivity. This stems from the fact that there is no agreement on what kind of formal representation is needed for an ontology. Dictionaries, taxonomies, thesauri and richly axiomatized formalizations can all be ontologies (Lehmann & Völker, 2014). Uschold and Grüninger (2004) introduced the continuum of ontologies, with very lightweight ontologies that consist of concepts only on one end of the spectrum, and formalized ontologies with relations between concepts being highly specified on the other end of the spectrum. Some knowledge representations have a richer meaning than others. The expressivity of ontologies relies mostly on the expressivity of the relation that is specified between concepts, as was also indicated by the ontology continuum.

Knowledge can be represented in many different ways. In section 2.1, it was represented using notation from set theory. It can also be represented with Resource Description Framework (RDF), which is a standardized model for knowledge representation (Jentzsch, Usbeck, & Vrandečić, 2014). One of the basic notions of RDF is a triple. Triples consist of three elements: a subject, a predicate and an object. The subject and object correspond

to the aforementioned concepts, while the predicate indicates the relation between these two concepts. An example of a triple is the following:

```
Sara leadResearcher grantProject
```

Triples can be expressed directly using RDF. We can create more triples, implicitly, by using more expressive semantics in addition to RDF. Languages that do this are, among others, RDFS and OWL (Jentzsch et al., 2014). RDFS can be used to describe class and property hierarchies. A class corresponds to what was called a concept earlier, and a property to a relation. An example of an RDFS triple is

```
leadResearcher rdfs:subPropertyOf researcher
```

Using the previous RDF triple and the RDFS triple, we can now infer that

```
Sara researcher grantProject
```

OWL expands the vocabulary to describe properties and classes even further. It allows classes to be defined on the basis of another class (e.g. complementOf) or multiple other classes (e.g. unionOf, intersectionOf). It is also possible to provide additional requirements for classes, (e.g. an instance of one class cannot be an instance of another class, using disjointWith). Properties can also be restricted using OWL vocabulary by specifying that, for example, at least one value of the property must come from a specific class (someValuesFrom).

The point of this subsection was to point the reader's attention to the fact that ontologies vary in their expressivity, and to illustrate what is meant by a highly specified formal model. This is important because the richness or expressiveness of ontology determines the complexity of learning one. Ontology learning is the subject of section 2.1.2.

2.1.2 Ontology Learning

Constructing an ontology is an expensive task which requires expert knowledge on both the domain of interest and ontology engineering. In order to create an ontology, the ontology engineer must acquire the specialized domain knowledge that needs to be contained in the ontology, and then formalize it in a manner that is logically sound. Once the ontology is constructed, it needs to be maintained, further increasing the costs of ontology engineering. While there is a great demand for structured knowledge representations, it is costly to do this manually, which is why techniques that can do this automatically using data are attractive. Ontology learning, a term that was coined by Mädche and Staab (2001), is concerned with this task.

Ontology learning hinges on the idea that domain knowledge is expressed in and can be extracted from text. When a text is written, the writer's

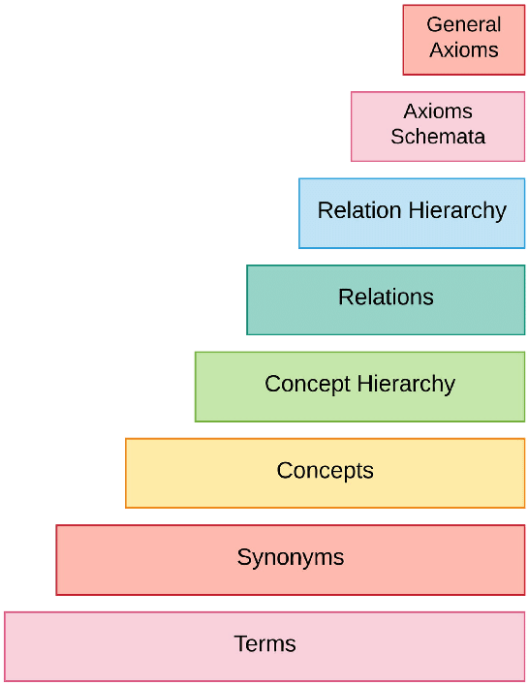

Figure 2.1: Ontology learning layer cake (reprinted from Asim et al. (2018))

knowledge of a domain is converted into text. That knowledge must then be reconstructed using the textual output of the writer. Asim, Wasim, Khan, Mahmood, and Abbasi (2018) refer to ontology learning as a reconstruction process. Crucial in this respect is that not all the writer's knowledge is explicitly transferred to text. In fact, Biemann (2005) claims that the more trivial or common-knowledge a piece of information is, the less likely it is to be lexicalized. This is a big (if not the biggest) limitation of ontology learning: only knowledge that is expressed in text can be extracted. Even with a perfect ontology learning method in place, knowledge that is not explicitly lexicalized can hardly or even impossibly be extracted. Cimiano (2014) states that an ontology reflects the conceptualization of a domain, whereas the results of an ontology learning algorithm reflects the idiosyncrasies of the dataset. He claims that this is the reason why "ontology learning has turned out to be much more difficult than expected" (p. v). It is important to be aware of the limitations of ontology learning. However, these limitations hold for many NLP tasks and may in part be solved by using more data. Despite its limitations, ontology learning is a valuable task.

A general approach to ontology learning is described by the ontology learning layer cake, a concept introduced by Buitelaar, Cimiano, and Magnini (2005). Figure 2.1 illustrates this concept. The first layer in the cake is term extraction, followed by synonym extraction. They are combined into

concepts, from which a hierarchy within the concepts is derived. RDFS vocabulary is well suited to express hierarchies. An example of a concept hierarchy expressed using RDFS is `mountainBike rdfs:subClassOf bike`. The fifth layer is relation extraction, and the sixth relation hierarchy extraction. A relation hierarchy can be expressed in the following way: `brotherOf rdfs:subPropertyOf siblingOf`. Next, axiom schemata are instantiated. An example of this step is to mark two classes as disjoint using OWL vocabulary. Finally, axioms are extracted from text. An example of an axiom is that every research project must have a lead researcher.

The ontology learning layer cake is not a normative model which indicates the steps that are to be followed in order to learn an ontology. It does give an overview of different aspects of ontology learning, and roughly indicates the complexity level of the task. The lower levels are relatively easy, whereas a higher level on the layer cake implies a more difficult task. The layer cake may be used as a roadmap, where the lower levels indicate the first steps, and the higher levels the final steps. However, this is not necessarily the case; certain levels may be collapsed or skipped altogether.

The layer cake must be viewed in relation to ontology expressivity. A highly expressive ontology requires all levels of the layer cake. Less expressive ontologies may lack concept and relation hierarchies, and axioms. Thus, the layer cake also represents a way to think about ontology learning in terms of ontology expressivity.

2.2 Information Extraction

In the previous section, we have seen what ontologies are, and what ontology learning entails. There is a great deal of variation in ontology expressivity, which is reflected in the complexity of the ontology learning task. In this thesis, I argue that information extraction is, at its core, quite similar to ontology learning. However, one must also realize that in order to create a fully fledged, highly expressive ontology, more is needed than information extraction might be able to provide, which we can understand better after seeing different types of ontologies. In section 2.2, information extraction is introduced, and in section 2.2.1, the connection between ontology learning and information extraction is established. In section 2.2.2, the state of the art in open information extraction is discussed.

Information extraction (IE) refers to extraction of structured representations from unstructured information in natural language (Jurafsky & Martin, 2009). These representations take the form of a tuple $\langle arg1, rel, arg2 \rangle$ containing two arguments and a semantic relation between them (Niklaus, Cetto, Freitas, & Handschuh, 2018). Given a sentence "Jeff founded Amazon", the tuple $\langle$ *Jeff, founded, Amazon* $\rangle$ should be extracted.

Early extraction tasks focused on the extraction of a specific argument: the named entity (Sarawagi, 2008). Over the years, information extraction has diversified and can now be applied more broadly to include not only named entities, but entities in general, or even more general arguments. Likewise, the extraction of relations has also opened up to more possibilities, most notably after the introduction of Open IE[1] by Banko, Cafarella, Soderland, Broadhead, and Etzioni (2007). Whereas traditional IE methods target a small set of predefined relations, Open IE extracts relations without the need to prespecify them.

2.2.1 Connection to Ontology Learning

In section 2.1.1, the ontology triple was introduced. It consists of a subject, predicate and an object. The information extraction triple consists of an arg1, relation and an arg2. Despite their different terminology, these concepts are interchangeable. The subject corresponds to the arg1, the predicate to the relation, and the object to the arg2. In this thesis, I use the terminology of the IE triple, since I will use IE methodology. Because the basic building block of both an ontology as well as an IE system is a triple, we can claim that the task of information extraction is very similar to the task of ontology learning. Ontology learning and information extraction are both concerned with extracting tuples containing concepts and a relation between them. However, for a highly expressive ontology, it is not enough to extract just these three instances and store them in a tuple. According to Suchanek (2014), there are three more requirements that need to be fulfilled in order to use this tuple in an ontology:

1. Canonicity: The entities must be disambiguated. In the example above, the information *founded Amazon* must be linked to the entity *Jeff Bezos* instead of another entity with the first name *Jeff*.

2. Taxonomic organization: The entities must be placed in a taxonomy. *Jeff Bezos* is a *founder*, which is a subclass of *person*. *Jeff Bezos* must not only be part of the ontology as a separate instance, but must be part of a taxonomy that indicates he is a *founder* and a *person*.

3. Consistency: The extracted information has to be consistent with the other information in the ontology. For example, the information *Jeff founded Amazon* is not consistent with an existing ontology that contains information that Amazon was founded in 1920 and that Jeff was born in 1970. In this case, the new information should not be entered until this inconsistency is resolved.

[1] In this thesis, I use the terms *information extraction* and *open information extraction* loosely: *IE* is also used to refer to Open IE. Since the field has evolved over the years, the distinction between the two has become less distinct. Moreover, the difference between the two is not of importance for this thesis.

These requirements must be seen in relation to ontology expressivity. Suchanek (2014) considers only highly expressive ontologies to be ontologies. Less expressive ontologies do not need to meet the requirements above. Again, though, it is important to be aware of the different forms of ontologies. Looking ahead, the goal of this thesis is to create a system by which triples for a less expressive ontology can be learned. Thus, the system created in this thesis will not meet the requirements set by Suchanek (2014).

The system will be created using techniques from open information extraction. Therefore, it is important to gain insight into the state of the art in open information extraction, which is described in section 2.2.2.

2.2.2 State of the Art

In order to determine the state of the art, systems need to be evaluated in the same ways, so they can be compared. The standard for Open IE evaluation is OIE2016 (Stanovsky & Dagan, 2016). Recently, a new standard has been proposed with the name CaRB (Bhardwaj, Aggarwal, & Mausam, 2019). The creators of this new standard argue that OIE2016 contains significant errors in the gold dataset and misses important tuples, hence a new standard for evaluation is needed. The two standards show that OLLIE, ClausIE, OpenIE4, OpenIE5, and PropS are among the best performing open IE systems. Table 2.1 summarizes the evaluation metrics of the most prominent open IE systems from both the CaRB and OIE2016 evaluation.[2] Overall, the results show that OpenIE4 has the best performance. In the following subsections, I describe the goals and methods of the systems from Table 2.1.

	CaRB				OIE2016
	Precision	Recall	F1	AUC	AUC
OLLIE	0.505	0.346	0.411	0.224	0.44 - 0.47
ClausIE	0.411	0.496	0.450	0.224	0.48 - 0.56
OpenIE4	0.553	0.437	0.488	0.272	0.53 - 0.56
OpenIE5	0.521	0.424	0.467	0.245	
PropS	0.340	0.300	0.319	0.126	0.44 - 0.46

Table 2.1: Evaluation metrics for different Open IE systems (adapted from Bhardwaj et al. (2019) and Stanovsky and Dagan (2016))

[2] OIE2016 assigns higher scores than CaRB due to a difference in scorers.

OLLIE

The goal of OLLIE (Mausam, Schmitz, Soderland, Bart, & Etzioni, 2012) was to improve on the state-of-the-art Open IE systems at the time, such as ReVerb (Fader, Soderland, & Etzioni, 2011) and WOE (Wu & Weld, 2010). It aims to overcome two weaknesses present in those systems: (1) only extracting relations expressed by verbs, and (2) ignoring context. OLLIE uses over 110,000 seed triples from ReVerb to bootstrap a large training set; it collects the sentences from a corpus that contain all the content words from the seed triple. Then, OLLIE learns open pattern templates (mappings from dependency paths to an extraction) using this training set. These patterns are used to match patterns in a sentence, yielding the desired triples. These triples are subjected to a context analysis. In this final step, information is added about the context of the triple, containing for example a condition under which the extraction is true.

ClausIE

ClausIE (Del Corro & Gemulla, 2013) is based on the idea that the clause type describes a minimal unit of coherent information in a clause, and that this can be used to form a proposition. Extracting coherent triples is an important challenge in Open IE, and clause types can be used to avoid incoherent triples. Examples of clause types are Subject Verb and Subject Verb Complement. ClausIE first computes a dependency parse for the sentence. Then, the clauses within the sentence are computed. Once the clauses are known, ClausIE determines the clause type (e.g. SV, SVO, SVOO, SVC) using the dependency pattern and information about the verbs. Finally, propositions are generated using those coherent clauses.

OpenIE4

OpenIE4[3] is a combination of SRLIE[4] (Christensen, Mausam, Soderland, & Etzioni, 2011) and RelNoun (Pal & Mausam, 2016). SRLIE generates tuples using Semantic Role Labeling (SRL). The first step of SRLIE is to process sentences with a dependency parser. Then, an SRL system is used to produce SRL frames. The final step is to process the frames to produce n-ary[5] extractions. This is necessary as many frame arguments are not considered extraction arguments in Open IE. The main purpose of using SRL frames is to be able to distinguish between information that is and is not asserted by the sentence. For example, the sentence "Early scientists believed that the earth is the center of the universe" (example from Mausam, 2016) does not assert that the earth is the center of the universe. Rather, it

[3]OpenIE is available at https://github.com/knowitall/openie
[4]SRLIE is available at https://github.com/knowitall/srlie
[5]These are needed for relations mediated by a ditransitive verb

asserts that early scientists believed this. Like OLLIE, SRLIE is devoted to solving the problem of extracting tuples that are not true, which is caused by ignoring context. RelNoun is a rule-based Open IE system that extracts noun-mediated relations, such as ⟨ Mlada, be bride of, Yaromir ⟩ from a sentence like "Mlada, Yaromir's bride,...". Extracting noun-mediated relations was the other challenge that OLLIE originally tried to solve.

OpenIE5

OpenIE5[6] is a combination of CALMIE (Saha & Mausam, 2018), BONIE (Saha, Pal, & Mausam, 2017), SRLIE and RelNoun. SRLIE and RelNoun are described above. CALMIE is specifically designed to improve processing of conjuctive sentences. It splits a conjunctive sentence into multiple simple sentences, and then passes these sentences to OpenIE4 (SRLIE and RelNoun). BONIE is designed to improve extraction from numerical sentences. It learns specific dependency patterns that express numerical relations by bootstrapping in a similar way OLLIE does.

PropS

PropS (Stanovsky, Ficler, Dagan, & Goldberg, 2016) outputs a representation of the propositional structure of a sentence. The proposition structure is induced from a (Stanford) dependency tree. The bigger bulk of the tree to proposition conversion is done in a rule-based fashion. Certain phenomena are difficult to identify from a dependency tree (such as the difference between raising and control). PropS uses heuristics to target these constructions and can then provide a fitting semantic representation.

2.3 Classification

In the previous section, information extraction was introduced and the state-of-the-art methods were highlighted. In this section, classification is introduced as a method for information extraction. First, a short introduction on classification is given. Then, several studies are reviewed in which classification is used for named entity recognition and relation extraction, two subtasks of information extraction.

Classification is a type of supervised learning where the aim is to categorize a set of data into classes. The algorithm used for classification in this thesis is a Support Vector Machine (SVM). Figure 2.2 illustrates how an SVM works. An SVM finds the optimal hyperplane[7] for linearly separable patterns. The optimal hyperplane is the hyperplane that maximizes the margin between the classes.

[6]OpenIE5 is available at https://github.com/dair-iitd/OpenIE-standalone

[7]A hyperplane is a higher-dimensional generalization of a line or plane

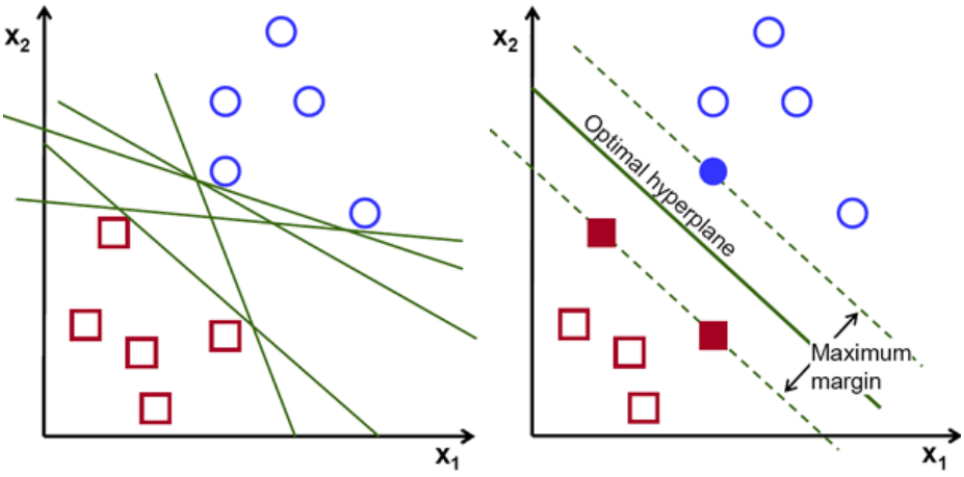

Figure 2.2: Support Vector Machine

SVMs can be extended to cases in which the data is not linearly separable in the following way. First, the data is transformed into higher dimensional data using a nonlinear mapping. This mapping is also known as a kernel function. Examples of kernels are the polynomial kernel or the radial basis function (RBF) kernel. After the transformation, a linear separating hyperplane is found in this new, higher-dimensional space. This last part is no different from what is done with linearly separable data. The difference is that the hyperplane that is found in the new space is a non-linear hyperplane in the original space.

2.3.1 Classification for information extraction

Classification techniques can and have been used for information extraction. This does require a reformulation of the problem; classification can be used to categorize a given input, not to extract entries that fit a certain class. I return to this issue when introducing my method in the next chapter. In what follows, I review studies concerned with two subtasks of information extraction: 1) named entity recognition, and 2) relation extraction.

Takeuchi and Collier (2002) create an SVM to classify named entities in the MUC-6 dataset and a molecular biology dataset. The features used are surface word forms, POS tags, orthographic features and word class tags of previous words. They found that POS tags hindered the performance on their biomedical data, most likely due to the fact that the tagger was trained on news texts. Their best-performing classifier obtains F1 scores of 0.718 and 0.748 on the different datasets.

Isozaki and Kazawa (2002) propose an SVM-based system to classify named entities in Japanese. POS tags, character types and the word itself

are collected for each word and its surrounding 2 words, yielding a total of 15 features. Their classifier achieves an F1 score of 0.900.

Ekbal and Bandyopadhyay (2008) create an SVM-based named entity recognizer for Bengali to classify named entities into four predefined categories: person, location, organization, and miscellaneous. They show that prefixes, suffixes, POS tags of current and surrounding 3 words, NE information of previous words, digit features and gazetteer lists are the features that contribute most to their NE system created for Bengali. They reach an F1 score of 0.918. Ekbal and Bandyopadhyay (2010) adapted the system to include Hindi, and incorporated some different features, including lexical context patterns.

Hong (2005) uses SVMs to detect and classify relations between entities in the ACE corpus. The features that were used include words, POS tags, (named) entity types, chunk tags, grammatical function tags, and the distance between two named entities. The best F1 score achieved for the combined task of detection and classification is .588. An F1 of .733 was achieved for classification only.

Choi and Kim (2013) propose a social relation extraction system that uses an SVM with a dependency kernel[8] to extract relations between two people mentioned in a text. The model with the best relation extraction performance reaches an F1 of 0.626.

Boella, Di Caro, and Robaldo (2013) create classifiers to extract semantic relations from legal texts. They create three binary SVMs; one for each semantic tag (active role, passive role, involved object). The syntactic dependencies of nouns are used as features. All nouns in a text are classified in one or none of the three semantic tags. The F1 score for classifying instances of active roles is relatively high (.948) compared to that of the passive role classifier (.423) and the involved object classifier (.414).

Torres, de Piñerez Reyes, and Bucheli (2018) present a semantic relations classifier for Spanish. Their aim is to detect and classify relations holding between two entities. The SVMs are trained on lexical, syntactic and semantic features. The model trained on these three types of features achieves an F1 score of .753.

In this section, examples were given of named entity recognition and relation extraction using a classification approach. However, classification has not been used to extract complete IE triples. In the next chapter, an experimental study is described that investigates this possibility.

[8]It is beyond the scope of this thesis to elaborate on dependency kernels. Suffice it to say that tree kernels are used to determine whether two entities are structurally related.

Chapter 3

Experimental Study

In chapter 2, ontology learning and information extraction were discussed. We saw that triples, i.e. tuples with three arguments[1]: relation, arg1 and arg2, are an important building block of both an ontology as well as an information extraction system. Classification techniques are used for information extraction, but not for ontology learning. Given the shared building block of ontologies and information extraction systems, I explore the possibility of using classification for ontology learning. The study presented in this chapter aims to answer the question if and how triples can be classified using a machine learning classifier. Answering this question is an important first step towards automatically creating ontologies using machine learning.

In section 2.3.1, examples were given of studies in which classification approaches were used for relation extraction and named entity recognition. Specifically, SVMs were used to classify relations or named entities. Therefore, I hypothesize that triples can also be classified with an SVM. Using classifiers for an extraction problem requires generating potential targets for classification, as a classifier predicts the class of a given input; it does not extract instances of a class. Thus, to extract triples using a classifier, the problem needs to be restated as a classification problem. This means potential triples need to be generated from a sentence, which can then be classified as either a triple or not a triple.

To answer the question *how* triples can be classified, several different models are created using different feature combinations. The three features used in this study are 1) bag of words, 2) bag of POS tags, 3) and bag of dependency parses. The IE studies using classification presented in section 2.3.1 often used features of a word and its neighboring words, including raw words or tokens, and POS tags. Since no features of neighboring words are collected in this study, dependency parses are used to take into account the context of a word (as a dependency is influenced by context).

[1]In this thesis, *arguments* is used to refer to the three arguments of a triple: relation, arg1, and arg2. When referring to the arguments of a relation, *arg1* and *arg2* are used.

The use of these features, and the classification approach in general, offers an important advantage over state-of-the-art Open IE methods such as the ones discussed in section 2.2.2. The method in this study does not rely on advanced NLP techniques such as semantic role labeling or context analysis. Rather, it uses basic tools like POS taggers and parsers, which are more likely to be available for languages other than English, and classifiers, which are language-independent.

In this thesis, two types of classifiers are made: a *triple classifier* and an *argument classifier*. The main goal of this thesis is to create an adequate triple classifier. The purpose of the three argument classifiers is two-fold. The first purpose is to investigate the performance of the triple classifier on all the three elements of a triple. By creating a separate relation classifier, for example, we can see whether an SVM with the features used in this thesis is an adequate method to classify a relation. If the three arguments cannot be classified properly, then classifying the complete triple properly is not realistic either. The opposite is not necessarily true: it is possible that the three arguments can be classified properly, but that the triple classifier does not have an adequate performance using the same method. This is because three items that are classified properly as arguments do not necessarily form a correct triple. The second purpose of the argument classifiers is for the triple classifier to be used on datasets without labeled triples in the future. For now, it suffices to say that argument classifiers are needed to generate potential arguments for the triple classifier if a dataset does not have labeled triples. The dataset used in this thesis does label instances of triples in a sentence. I return to the second purpose in section 4.2.

The structure of this chapter is as follows. Section 3.1 details the method for creating the classifiers. First, the dataset is described. Second, the algorithm is discussed. This entails feature extraction, preprocessing, and training and testing procedures. Section 3.2 gives the results of the study. Section 3.3 is a discussion of those results.

3.1 Method

This section lays out the method of creating the argument classifiers and triple classifier. In section 3.1.1, I describe and illustrate the data that is used to train and test the classifiers. First, the CaRB dataset is described. This dataset contains sentences and accompanying triples, but no negative examples of triples. In the second part of section 3.1.1, I describe how the negative examples were generated to create training and testing data. Section 3.1.2 describes the algorithm used to create the classifiers. First, feature extraction is described. Second, I detail what models are created and how that is done. This includes a description of different feature combinations that were used, preprocessing steps and and the training and testing process.

3.1.1 Dataset

CaRB dataset

The CaRB dataset (Bhardwaj et al., 2019) is used to train the classifiers. The dataset consists of over 1,200 sentences from which 5,000 triples are extracted by human annotators. The sentences are sampled from English Wikipedia and the CoNLL-2009 shared task (Bhardwaj et al., 2019; Hajič et al., 2009; He, Lewis, & Zettlemoyer, 2015; Stanovsky & Dagan, 2016). As an example, a sentence with three triples is taken from the dataset:

Voyslava has killed Mlada , Yaromir 's bride , to have him for herself .
⟨ has killed, Voyslava, Yaromir 's bride ⟩
⟨ has killed, Voyslava, Mlada ⟩
⟨ is bride of, Mlada, Yaromir ⟩

Triples are tuples consisting of three arguments: a relation (e.g. *has killed*), the relation's first argument (arg1, e.g. *Voyslava*), and its second argument (arg2, e.g. *Yaromir's bride*). In the CaRB dataset, the majority of tuples has three arguments, such as the examples given above. However, there are tuples with a different number of arguments. In this thesis, only tuples with three arguments are used, as a fixed number of arguments is preferable for the purposes of training a classifier, and triples are the canonical tuples. Tuples with more than three arguments are adapted to triples, and tuples with two arguments are removed from the dataset.

Tuples with more than three arguments consist of a relation, an arg1, an arg2, and one or more modifiers. The entries with modifiers are preserved, because the modifiers can be removed, yielding a tuple with three arguments. An example of an entry with a modifier is given below:

Mr. Baker found an opening under the house .
⟨ found, Mr. Baker, an opening, under the house ⟩

An entry with a ditransitive verb is not an example of a tuple with more than three arguments. Ditransitive verbs are captured in two triples in the CaRB dataset. An example is given below:

XM offers a free month of service to subscribers who called in complaints
⟨ offers, XM, a free month of service ⟩
⟨ offers to, XM, subscribers who called in complaints ⟩

Second, there are tuples with only two arguments. Although it is not frequent in the dataset, there are intransitive verbs and transitive verbs that spell out only one argument[2]. Entries that have a relation with only one argument are removed from the dataset. An example is given below:

Competition in the sale of complete bikes is heating up too .
⟨ is heating up too, Competition in the sale of complete bikes ⟩

[2]In this case, *argument* refers to the arguments of a verb instead of the arguments of a triple. In the remainder of this thesis, it does refer to the arguments of a triple.

Creating the training and testing data

The CaRB dataset contains only positive examples of triples that are extracted from a sentence. In order to train a classifier, negative examples are also necessary. First, I explain how the training and testing data for the arg1, arg2 and relation classifiers is created. Then, I explain how the training and testing data for the triple classifier is created.

Argument classifier data The dataset contains positive examples of triples. The arguments of these triples are the positive examples of arguments. Negative examples of arguments are generated using the characteristics of the positive examples to minimize differences between positive and negative examples in the data. The lengths (number of tokens) of all positive examples of an argument are collected. This is done by creating a list for each argument (arg1, arg2 and relation) containing the lengths of all the examples of that argument. Say one example of an arg1 contains two tokens, another one contains seven tokens, and yet another contains three, then a list `[2,7,3]` is created. For each positive example of an argument, a negative example is generated, randomly choosing a length from the list gathered to determine the number of tokens the argument must consist of. Suppose the sentence is "11 million copies of the flyer were distributed to the public", and the length that was randomly chosen from the list containing arg2 lengths is 2. In that case, two tokens are randomly chosen from the sentence, yielding the negative arg2 example "public distributed".

	Example
sentence	11 million copies of the flyer were distributed to the public
relation	were distributed
arg1	11 million copies of the flyer
arg2	to the public
neg relation	copies to million
neg arg1	of the flyer
neg arg2	public distributed

Table 3.1: Example of training and testing data

After the negative training and testing data is generated, it is added to the positive training and testing data. The result of merging the positive and negative data is indicated by *training and testing data (arguments)* in Figure 3.1. The negative examples of arguments taken from a given sentence are displayed on one row, and labeled with a 0.

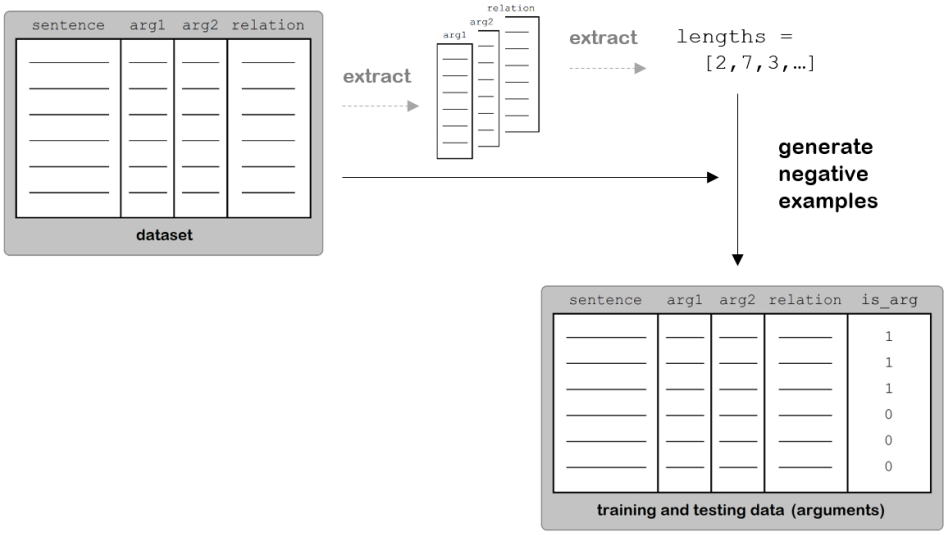

Figure 3.1: Creating training and testing data for the argument classifier

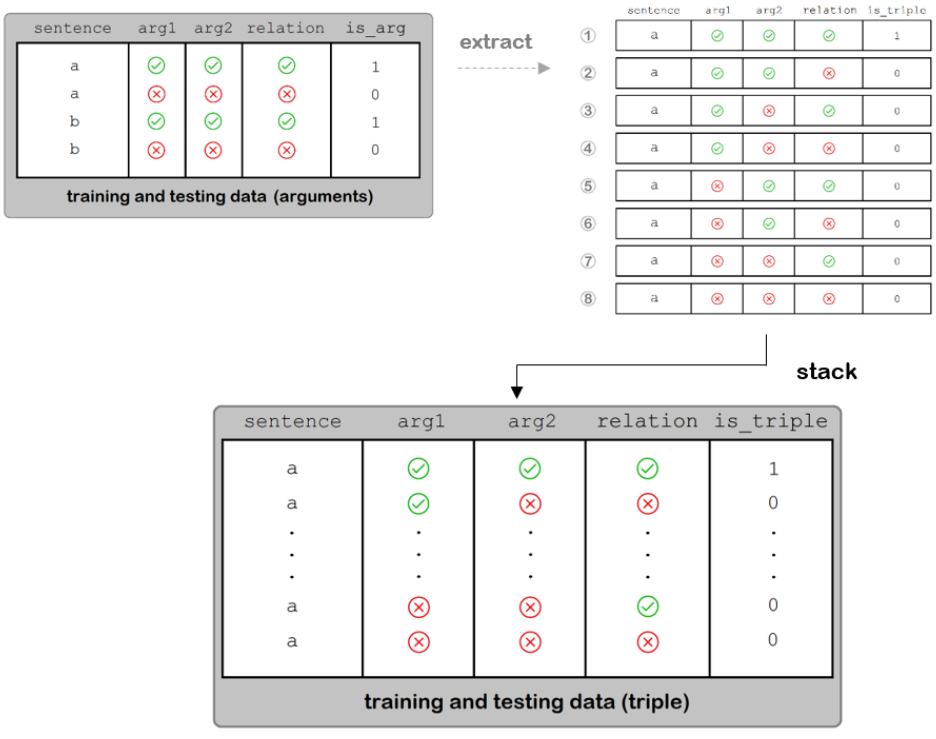

Figure 3.2: Creating training and testing data for the triple classifier

Triple classifier data To create a triple classifier, training and testing data is needed that lists positive and negative examples of triples. The training and testing data created for the argument classifiers does not suffice for the purposes of training and testing a triple classifier, as it lists positive and negative examples of arguments. The negative training and testing data for the triple classifier should consist of a combination of positive and negative examples of arguments; it should not be only negative arguments. However, the arguments training and testing data is used as a starting point for the triple training and testing data. Figure 3.2 shows the process of creating training and testing data for the triple classifier. The *training and testing data (arguments)* corresponds to that in Figure 3.1. Creating the triple training and testing data is done by extracting all possible combinations of positive and negative arguments for a sentence. In Figure 3.2, a positive argument is indicated with a tick and a negative argument is indicated with a cross. It can be seen that a sentence (e.g. *a* or *b*) has an entry with a positive set of arguments and an entry with a negative set of arguments. There are $2^3 = 8$ triples generated for each pair of a positive and a negative entry. One of these triples is a positive example of a triple (only positive arguments), and the other 7 are negative examples of triples (ranging from 1 negative argument to only negative arguments). The generated negative triples are stacked and added to the positive triples, creating the *training and testing data (triple).*

3.1.2 Algorithm

Feature Extraction

The training and testing data that was described in the previous section contains sentences and the arguments making up a triple. More features are generated for the classifier to train on. These features are extracted using the data present in the training and testing data. The features used in this study are words, POS tags and dependency parses. Figure 3.3 details the steps of the feature extraction process: extracting sentences, tagging and parsing them, extracting arguments, collecting the argument's POS tags and dependency parses, cleaning the tags and parses, concatenating them and adding them to the training and testing data to create training and testing data with features.

First of all, the sentences are extracted from the dataset, after which they are tagged and parsed using SpaCy (Honnibal & Montani, 2017). Then, the arguments are extracted, and the POS tags and dependency parses are collected for each argument token. In Figure 3.3, the sentence "the woman walked the dog" is used as an example. Suppose the triple belonging to this sentence is ⟨ walked, the woman, the dog ⟩. POS tagging and dependency parsing is done on the sentence, the arguments (relation, arg1, or arg2) are

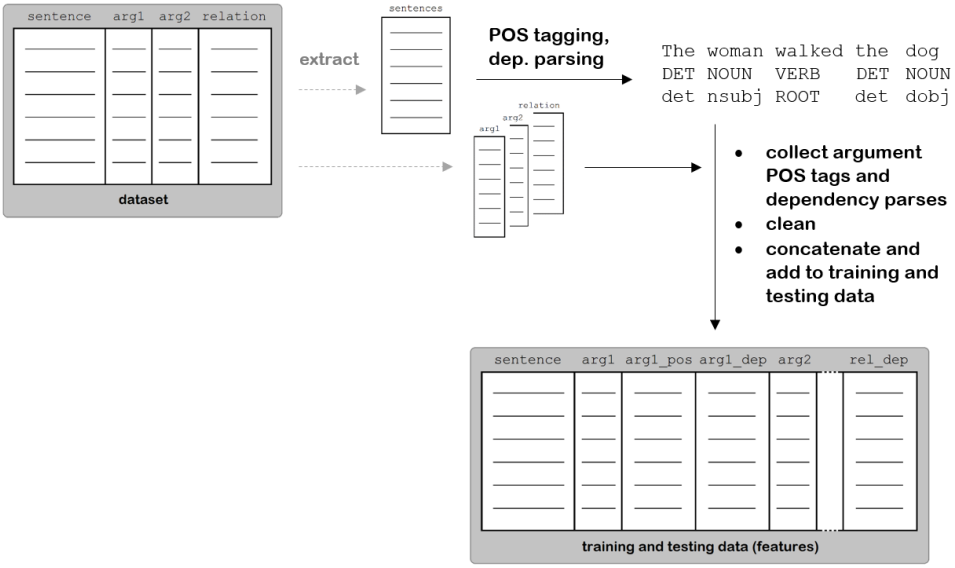

Figure 3.3: Creating features for the training and testing data

extracted from the dataset and their POS tags and dependency parses are collected. Arg1 consists of two tokens: "the" and "woman". Two POS tags are collected for "the", which are both associated with the *DET* tag. "Woman" is associated with the *NOUN* tag. Two dependency parses are collected for "the", both of with are *det*. The dependency parse collected for "woman" is *nsubj*.

The difficulty is to make sure that the argument token corresponds to the correct token in the sentence in case a token occurs more than once.[3] That is why first all the POS tags and dependency parses are collected, and then an additional step of cleaning is done. In this cleaning step, it is checked whether tokens have multiple POS tags or dependency parses. This is the case when a token occurs twice in the sentence. Suppose a token occurs twice and it has two different POS tags. In this case, the token's POS tags are removed. The rationale behind this is to prevent noise. No POS tag of a certain token is chosen instead of a correct POS tag and one or more false ones. If the multiple POS tags are identical, the redundant POS tags are removed. This would have been done in the example used before, where both instances of "the" in the sentences were associated with the *DET* tag. The same process is used to check if a token has multiple dependency parses.

Besides checking for multiple POS tags and dependency parses, the cleaning step incorporates a check for implicit sentences. It is checked whether all the tokens in the triple are present in the sentence. If they

[3]The position of an argument's token in the sentence is not indicated in the original CaRB dataset. That is likely due to the fact that not all tokens in the triple's argument are part of the sentence.

are not, the triple is considered implicit. An example of an implicit triple is given below:

Voyslava has killed Mlada , Yaromir 's bride , to have him for herself .
⟨ is bride of, Mlada, Yaromir ⟩

It can be gathered from "Mlada, Yaromir's bride" that Mlada is the bride of Yaromir. This is captured in the triple given in the example. I consider the relation implicit, as the tokens from "is bride of" are not present in the sentence. If one of the triple's arguments is implicit, then the triple is implicit. Implicit triples are removed from the dataset for two reasons: 1) their POS tags and dependency parses cannot be collected from the sentence, and 2) to minimize differences between the positive and negative examples of triples. Since negative examples are generated using only tokens from the sentence, it is fairer to allow only positive examples with tokens that are present in the sentence.

After collecting and cleaning the POS tags and dependency parses, the POS tags and dependency parses of all argument tokens are concatenated and added to the training and testing data. An example of the data yielded by the process is given in Table 3.2 and is indicated by *training and testing data (features)* in Figure 3.3.

	Example
sentence	11 million copies of the flyer were distributed to the public
relation	were distributed
arg1	11 million copies of the flyer
arg2	to the public
relation POS	AUX VERB
arg1 POS	NUM NUM NOUN ADP D NOUN
arg2 POS	ADP D NOUN
relation dep	auxpass ROOT
arg1 dep	compound nummod nsubjpass prep det pobj
arg2 dep	prep det pobj

Table 3.2: Example of training and testing data with features

Models

A support vector machine (SVM) with an RBF kernel was trained on the triple data to create a triple classifier. Separate SVMs with RBF kernels were trained on the argument data to create three argument classifiers. For all classifiers, 2-fold cross-validation[4] was used to train and test the models, since a limited data sample is available. The training and testing data was vectorized again in each fold to create a model that is based on the training data only, and is not influenced by the testing data. Vectors with absolute counts were used. Different models were created by training the classifiers on different combinations of three features: words, POS tags and dependency parses. When a classifier is trained using a feature x, the training and testing data are vectorized, creating a bag of x representation. The following models were created for each of the four classifiers:

- Bag of Words (**BoW**): a BoW model is created by training the classifier on the vectorized columns with the original arguments or triples, containing the tokens or words that make up the argument or triple.
- Bag of POS tags (**BoP**): a BoP model is created by training the classifier on the vectorized POS tags of the argument or triple.
- Bag of dependency parses (**BoD**): a BoD model is created by training on the dependency parses.
- Combination of BoW and BoP (**BoW+BoP**): this model is created by training on both the word feature as well as the POS tag feature.
- Combination of Bow, BoP and BoD (**BoW+BoP+BoD**): this model combines all the previous models, as it is created by training on all the available features.

A baseline is created by randomly predicting a binary classification. The arguments have a 0.5 chance of being classified correctly, as there are two classes that are evenly distributed. The chance of a triple being classified correctly is lower. This is due to the distribution of positive and negative examples of triples: one out of eight potential triples is actually a triple. Since the prediction is done randomly, many triples are misclassified.

[4] A number of models were also created using 5-fold and 10-fold cross-validation, in order to examine whether there was a performance difference. After all, a model trained with 2-fold cross-validation has only seen 50% of the data, whereas a model trained with 10-fold cross-validation is trained on 90% of the data. The mean scores of models trained using 2-, 5- and 10-fold cross-validation were comparable. This indicates that the classifiers can be trained on a small amount of data, as the performance does not change considerably when using a smaller portion of the data is used.

3.2 Results

The performance of the four classifiers - arg1, arg2, relation, and triple - is given in this section. Tables 3.3, 3.4 and 3.5 show the precision, recall and F1 of the different models created using different features. The baseline model that was created shows a better performance for the argument classifiers than for the triple classifier, as expected. The BoW model shows considerable improvement on the baseline for the argument classifiers, but not for the triple classifier. The BoP and BoD model do show an increase in performance compared to the baseline for all classifiers. With two minor exceptions in the recall, the BoW+BoP+BoD model, which combines all three features, is the best performing model. The F1 of the relation classifier is the highest (0.916), followed by the arg1 classifier (0.887) and the arg2 classifier (0.797). The triple classifier reaches an F1 of 0.520, which is considerably higher than the baseline.

	baseline	BoW	BoP	BoD	BoW+ BoP	BoW+ BoP+ BoD
arg1	0.433	0.782	0.743	0.875	0.832	**0.890**
arg2	0.520	0.621	0.701	0.751	0.726	**0.767**
relation	0.552	0.796	0.854	0.863	0.885	**0.908**
triple	0.084	0.200	0.395	0.482	0.454	**0.542**

Table 3.3: Precision of the random prediction baseline, bag of words (BoW), bag of POS tags (BoP), bag of dependency parses (BoD), BoW and BoP combined, and BoW, BoP and BoD combined

	baseline	BoW	BoP	BoD	BoW+ BoP	BoW+ BoP+ BoD
arg1	0.542	0.637	0.862	0.839	0.865	**0.883**
arg2	0.448	0.481	0.774	0.803	0.733	**0.830**
relation	0.593	0.701	0.894	0.881	**0.926**	0.925
triple	0.360	0.164	0.457	**0.507**	0.412	0.501

Table 3.4: Recall of the random prediction baseline, bag of words (BoW), bag of POS tags (BoP), bag of dependency parses (BoD), BoW and BoP combined, and BoW, BoP and BoD combined

	baseline	BoW	BoP	BoD	BoW+ BoP	BoW+ BoP+ BoD
arg1	0.481	0.702	0.798	0.856	0.848	**0.887**
arg2	0.481	0.542	0.736	0.776	0.729	**0.797**
relation	0.571	0.745	0.873	0.872	0.905	**0.916**
triple	0.136	0.181	0.424	0.495	0.432	**0.520**

Table 3.5: F1 of the random prediction baseline, bag of words (BoW), bag of POS tags (BoP), bag of dependency parses (BoD), BoW and BoP combined, and BoW, BoP and BoD combined

3.3 Discussion

The main goal of this thesis is to answer the question how ontology triples can be learned using a machine learning classifier. The results indicate that the triple classifier which was trained on word, POS and dependency features has the best performance. Given the performance of the different models, the most important feature seems to be the dependency parses. Of the models containing only one feature, the dependency model (BoD) has the best performance. Moreover, the model with two combined features (BoW+BoP) improves considerably when the dependency feature is added. This might be because bag of x models cannot take word order into account. Adding a dependency feature is a way of taking syntactic information into account, which is (partly) expressed by word order in English.

Compared to the state of the art open information extraction systems, discussed in section 2.2.2, the best model created in this thesis performs well. The best performing open information extraction tool is OpenIE4, which reaches an F1 of 0.488. The performing model in this thesis is the BoW+BoP+BoD model, which achieves an F1 of 0.520. However, the results of OpenIE4 and the classifiers presented in this thesis cannot readily be compared, as the tasks are somewhat different. In the state of the art methods, all triples are generated, positive and negative ones. In this thesis, the positive triples were taken from the dataset, and the negative triples were generated. As such, there was no chance that the correct positive example was not generated. Thus, while the performance scores of the classifiers in this study may be higher, the task was also somewhat easier.

As mentioned in the introduction of this chapter, one of the purposes of the argument classifiers was to offer an insight into the performance of the triple classifier. We can conclude that whatever might be lacking in the performance of the triple classifier is not due to an inability to classify arguments. Regarding the argument classifiers, the relation classifier obtains the

best results, followed by the arg1 and the arg2 classifier. An ad hoc analysis was done to investigate whether negative instances of certain arguments were more frequent in misclassified triples. For example, did misclassified triples more often contain a negative arg2 than a negative relation? The analysis revealed no particular patterns, suggesting that the performance of the argument classifiers does not directly translate into the performance of the triple classifier. The false positives and false negatives were also investigated to check if certain feature patterns were often misclassified (e.g. relations with prepositions were often misclassified). Again, no patterns stood out as being misclassified more often than others.

Mapping the results from this study to the ontology learning layer cake introduced in section 2.1.2 is not straightforward. The scope of this study was term/concept extraction and relation extraction. The layer cake presupposes, however, that the relations are extracted using certain concepts. That was not the case in this study, which means that the performance of those two tasks are independent. Moreover, the performance of the argument classifiers revealed that relation extraction was more successful than concepts (arg1 and arg2) extraction. Thus, the method of this study might succeed on tasks higher in the layer cake, while being less successful at supposedly easier tasks.

Chapter 4

Discussion

4.1 Conclusion

In this thesis, I investigated the possibility of using machine learning classifiers to classify triples for an ontology. I argued that ontology triples and information extraction triples are interchangeable with respect to their structure, and used this insight to transfer methods from open information extraction to ontology learning. Up to date, classifiers have been used for subtasks of information extraction, such as relation extraction and named entity recognition. The results show the approach taken in this thesis is promising. The argument classifiers have a relatively high performance, as expected, given that classifiers have been used for these purposes in the past. The best-performing triple classifier is an SVM trained on word, POS and syntactic dependency features. It obtains an F1 score of 0.520, which outperforms the state of the art open information extraction systems, albeit on a slightly easier task. These results indicate that the novel approach taken in this thesis is promising.

4.2 Future Research

The most obvious question for further research is how the triple classifier can be improved and if this can be done up to a point where the classifier can be used to populate an ontology with. Regarding the first part of that question, it is worth exploring new features that contain information about how arguments are related to each other in a potential triple. To give an example, a feature explicitly indicating whether there is a syntactic dependency between the arg1 and the relation might prove useful. The reason to assume it might be is because the argument classifiers perform well compared to the triple classifier: while the classifiers are adequate at categorizing an argument, classifying whether these arguments belong together in a triple is not as straightforward. Thus, any information indicating whether a set of argu-

ments form a triple might improve the triple classifier. Regarding the second part of the question, it is reasonable to assume that a classifier approach will always contain some wrongly classified items. However, for the purposes of populating an ontology, a classifier approach could provide an adequate enough solution. First of all, any mistakes contained in the ontology could be solved by an ontology engineer. Second, this classifier approach offers a low entry point, which is especially interesting when extracting an ontology for other languages other than English, for which less resources are available. All that is needed is a dataset, a POS tagger and parser. Compared to the requirements of the state of the art methods, which include context analysis, clause computation, semantic role labeling and more, that is not much. Finding a usable dataset is the most difficult, although this thesis shows that a smaller dataset is also adequate. Compared to state-of-the-art open information extraction systems, the resources used in this thesis are more likely to be available for languages other than English.

Another interesting question is how well the results hold up when all triples are generated. Much is already in place to generate all triples. The argument classifiers could be used to filter potential arguments. The triples could be generated using different combinations of potential arguments, in the same way that the negative triples were generates in this study. The reason not all triples were generated in this thesis is that the number of generated triples would have been quite explosive. This in turn requires a way to trim down the amount of triples generated. Figuring out how to do this best was beyond the scope of this thesis. That is why the option was chosen to use the positive triples available. If all triples are to be generated, I expect especially recall to be somewhat lower, since there is a chance that the positive triple is not generated at all. However, since the performance of the argument classifier is quite good, generating potential triples using these classifiers is a promising topic of further research.

References

Asim, M. N., Wasim, M., Khan, M. U. G., Mahmood, W., & Abbasi, H. M. (2018). A survey of ontology learning techniques and applications. *Database*, *2018*, 1-24. Retrieved from https://doi.org/10.1093/database/bay101

Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., & Etzioni, O. (2007). Open Information Extraction from the Web. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence* (p. 2670–2676). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Bhardwaj, S., Aggarwal, S., & Mausam. (2019). CaRB: A Crowdsourced Benchmark for Open IE. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 6262–6267). Hong Kong, China: Association for Computational Linguistics. Retrieved from https://doi.org/10.18653/v1/D19-1651

Biemann, C. (2005). Ontology Learning from Text: A Survey of Methods. *LDV Forum*, *20*(2), 75-93.

Boella, G., Di Caro, L., & Robaldo, L. (2013). Semantic Relation Extraction from Legislative Text Using Generalized Syntactic Dependencies and Support Vector Machines. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web* (pp. 218–225). Heidelberg, Germany: Springer. Retrieved from https://doi.org/10.1007/978-3-642-39617-5_20

Borst, W. (1997). *Construction of Engineering Ontologies for Knowledge Sharing and Reuse* (PhD thesis). Institute for Telematica and Information Technology, University of Twente.

Buitelaar, P., Cimiano, P., & Magnini, B. (2005). Ontology Learning from Text: An Overview. In P. Buitelaar, P. Cimiano, & B. Magnini (Eds.), *Ontology Learning from Text: Methods, Evaluation and Applications* (p. 3-12). Amsterdam, The Netherlands: IOS Press.

Choi, M., & Kim, H. (2013). Social relation extraction from texts using a support-vector-machine-based dependency trigram kernel. *Information Processing and Management*, *49*(1), 303–311. Retrieved from

https://doi.org/10.1016/j.ipm.2012.04.002

Christensen, J., Mausam, Soderland, S., & Etzioni, O. (2011). An Analysis of Open Information Extraction Based on Semantic Role Labeling. In *Proceedings of the Sixth International Conference on Knowledge Capture* (p. 113–120). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1999676.1999697

Cimiano, P. (2014). Foreword. In J. Lehmann & J. Völker (Eds.), *Perspectives on Ontology Learning* (pp. v–viii).

Del Corro, L., & Gemulla, R. (2013). ClausIE: Clause-Based Open Information Extraction. In *Proceedings of the 22nd International Conference on World Wide Web* (p. 355–366). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2488388.2488420

Ekbal, A., & Bandyopadhyay, S. (2008). Bengali Named Entity Recognition using Support Vector Machine. In *Proceedings of the IJCNLP-08 Workshop on Named Entity Recognition for South and South East Asian Languages* (pp. 51–58).

Ekbal, A., & Bandyopadhyay, S. (2010). Named Entity Recognition using Support Vector Machine: A Language Independent Approach. *International Journal of Electrical, Computer, and Systems Engineering*, *4*(2), 155–170. Retrieved from https://doi.org/10.5281/zenodo.1057979

Fader, A., Soderland, S., & Etzioni, O. (2011). Identifying Relations for Open Information Extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing* (pp. 1535–1545). Edinburgh, Scotland, UK: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/D11-1142

Gruber, T. R. (1995). Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human - Computer Studies*, *43*(5-6), 907–928. doi: 10.1006/ijhc.1995.1081

Gruber, T. R. (2008). Ontology. In L. Liu & M. T. Özsu (Eds.), *Encyclopedia of Database Systems* (pp. 1963–1965). Springer-Verlag.

Guarino, N., Oberle, D., & Staab, S. (2009). What Is an Ontology? In S. Staab & R. Studer (Eds.), *Handbook on Ontologies* (pp. 1–17). Springer-Verlag.

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M. A., Màrquez, L., ... Zhang, Y. (2009). The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task* (pp. 1–18). Boulder, Colorado, USA: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/W09-1201

He, L., Lewis, M., & Zettlemoyer, L. (2015). Question-Answer Driven Semantic Role Labeling: Using Natural Language to Annotate Natural Language. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 643–653). Lisbon, Portugal: Association for Computational Linguistics. Retrieved from https://doi.org/10.18653/v1/D15-1076

Hong, G. (2005). Relation Extraction Using Support Vector Machine. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing* (pp. 366–377).

Honnibal, M., & Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. *To appear*.

Isozaki, H., & Kazawa, H. (2002). Efficient Support Vector Classifiers for Named Entity Recognition. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1* (p. 1–7). Taipei, Taiwan: Association for Computational Linguistics. Retrieved from https://doi.org/10.3115/1072228.1072282

Jentzsch, A., Usbeck, R., & Vrandečić, D. (2014). An Incomplete and Simplifying Introduction to Linked Data. In J. Lehmann & J. Völker (Eds.), *Perspectives on Ontology Learning* (pp. 21–33).

Jurafsky, D., & Martin, J. H. (2009). *Speech and Language Processing* (2nd ed.). London, England: Pearson Education.

Lehmann, J., & Völker, J. (2014). An Introduction to Ontology Learning. In J. Lehmann & J. Völker (Eds.), *Perspectives on Ontology Learning* (pp. ix–xvi).

Mausam. (2016). Open Information Extraction Systems and Downstream Applications. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (p. 4074–4077). AAAI Press. Retrieved from https://dl.acm.org/doi/10.5555/3061053.3061220

Mausam, Schmitz, M., Soderland, S., Bart, R., & Etzioni, O. (2012). Open Language Learning for Information Extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning* (pp. 523–534). Jeju Island, Korea: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/D12-1048

Mädche, A., & Staab, S. (2001). Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, *16*(2), 72–79.

Niklaus, C., Cetto, M., Freitas, A., & Handschuh, S. (2018). A Survey on Open Information Extraction. In *Proceedings of the 27th International Conference on Computational Linguistics.* Santa Fe, New Mexico, USA: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/C18-1326

Pal, H., & Mausam. (2016). Demonyms and Compound Relational Nouns in Nominal Open IE. In *Proceedings of the 5th Workshop on Automated Knowledge Base Construction* (pp. 35–39). San Diego, CA, USA: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/W16-1307

Saha, S., & Mausam. (2018). Open Information Extraction from Conjunctive Sentences. In *Proceedings of the 27th International Conference on Computational Linguistics* (pp. 2288–2299). Santa Fe, New Mexico, USA: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/C18-1194

Saha, S., Pal, H., & Mausam. (2017). Bootstrapping for Numerical Open IE. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (p. 317-323). Vancouver, Canada: Association for Computational Linguistics. Retrieved from https://doi.org/10.18653/v1/P17-2050

Sarawagi, S. (2008). Information Extraction. *Foundations and Trends in Databases*, *1*(3), 261–377. Retrieved from https://doi.org/10.1561/1900000003

Stanovsky, G., & Dagan, I. (2016). Creating a Large Benchmark for Open Information Extraction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (pp. 2300–2305). Austin, Texas, USA: Association for Computational Linguistics. Retrieved from https://doi.org/10.18653/v1/D16-1252

Stanovsky, G., Ficler, J., Dagan, I., & Goldberg, Y. (2016). Getting More Out Of Syntax with PropS. *CoRR*. Retrieved from http://arxiv.org/abs/1603.01648

Suchanek, F. (2014). Information Extraction for Ontology Learning. In J. Lehmann & J. Völker (Eds.), *Perspectives on Ontology Learning* (pp. 135–151).

Takeuchi, K., & Collier, N. (2002). Use of Support Vector Machines in Extended Named Entity Recognition. In *Proceedings of the 6th Conference on Natural Language Learning* (p. 1–7). Stroudsburg, Pennsylvania, USA: Association for Computational Linguistics. Retrieved from https://doi.org/10.3115/1118853.1118882

Torres, J. P., de Piñerez Reyes, R. G., & Bucheli, V. A. (2018). Support Vector Machines for Semantic Relation Extraction in Spanish Language. In *Proceedings of the 13th Colombian Conference on Computing 2018* (pp. 326–337). Retrieved from https://doi.org/10.1007/978-3-319-98998-3_26

Uschold, M., & Grüninger, M. (2004). Ontologies and Semantics for Seamless Connectivity. *SIGMOD Record*, *33*(4), 58–64. Retrieved from https://doi.org/10.1145/1041410.1041420

Wu, F., & Weld, D. S. (2010). Open Information Extraction Using Wikipedia. In *Proceedings of the 48th Annual Meeting of the Asso-*

ciation for Computational Linguistics (pp. 118–127). Uppsala, Sweden: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/P10-1013