PRIFYSGOL
BANGOR
UNIVERSITY

School of Computer Science and Electronic Engineering
College of Environmental Sciences and Engineering

A Novel Python Tool for Information Extraction of Swedish Text

---

Dan Stoakes

Submitted in partial satisfaction of the requirements for the
Degree of Bachelor of Science
in Computer Science

Supervisor Dr. W. J. Teahan

May 2021

Acknowledgements

> *Remembering that you are going to die is the best way I know to avoid the trap of thinking you have something to lose.*
>
> — **Steve Jobs**

I would like to thank everybody who has supported and helped me through my dissertation. All members of the Computer Science faculty have been wonderful throughout my time at Bangor and have my thanks. In particular, I would like to thank my supervisor, Dr. William Teahan, for respecting and understanding my slightly haphazard work-style, as well as offering greatly appreciated feedback regarding my work.

Statement of Originality

The work presented in this thesis/dissertation is entirely from the studies of the individual student, except where otherwise stated. Where derivations are presented and the origin of the work is either wholly or in part from other sources, then full reference is given to the original author. This work has not been presented previously for any degree, nor is it at present under consideration by any other degree awarding body.

Student:

Dan Stoakes

Statement of Availability

I hereby acknowledge the availability of any part of this thesis/dissertation for viewing, photocopying or incorporation into future studies, providing that full reference is given to the origins of any information contained herein. I further give permission for a copy of this work to be deposited with the Bangor University Institutional Digital Repository, and/or in any other repository authorised for use by Bangor University and where necessary have gained the required permissions for the use of third party material. I acknowledge that Bangor University may make the title and a summary of this thesis/dissertation freely available.

Student:

Dan Stoakes

Abstract

Information Extraction (IE) is the automated retrieval of specific, pre-defined information from natural language text. The information extraction task is a language-agnostic process which represents a component of Natural Language Processing (NLP). Swedish NLP has seen few developments in recent years, particularly for IE, despite the language featuring a growing prevalence on the internet.

This dissertation explores the relevant literature for IE and the existing resources which are available in modern Swedish NLP. A Swedish information extraction implementation in the form of a novel Python tool is also presented, along with a comprehensive evaluation of the implementation. The results shown within this dissertation show promise for the advancement of Swedish NLP, with some results showing improvement over existing systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Information Extraction is a core aspect of Natural Language Processing, with applications in medicine, document curation, database population, and military operations [47] [41] [37]. IE has an increased significance in the modern age because of the large, ever-growing amount of information available through the internet. This information contains raw, unstructured, and unfiltered natural language text which is useful for various NLP tasks. One nation which features a particularly high internet usage rate is Sweden, with 98% of the population using the internet in 2021. This ranks Sweden as the third highest country in the world for internet penetration [25].

Despite such a high internet usage rate, Sweden has relatively few NLP resources. This is particularly true for Information Extraction, with no IE systems listed on Språkbanken, Sweden's leading linguistic research unit (see Chapter 2). Other linguistic resources, such as corpora, are also relatively sparse, despite an influx of Swedish information production through the internet. Furthermore, standard NLP libraries and packages, such as NLTK [6] and spaCy, lack any native Swedish support.

The combination of high internet usage and relatively few NLP resources highlights the need for further developments within Swedish NLP. Various elements of the NLP pipeline have implementations, such as Swedish spaCy, Stagger [33], etc., but there is a definite need for a general-purpose Swedish IE system.

1.2 Aims and Objectives

The overall aim of this dissertation is:

- To create a novel tool which implements a method for information extraction of Swedish text using Python.

The specific objectives of the research are:

1. To review the relevant literature in the Natural Language Processing field, with specific reference to Information Extraction. (See Chapter 2).

2. To implement Information Extraction with Swedish text as a command-line Python tool. (See Chapters 3 and 4).

3. To compare the implementation and results against the training set. (See Chapter 5).

1.3 Dissertation Summary

- Chapter 1 – Introduction: Overview of the background and motivation behind Information Extraction.

- Chapter 2 – Literature Review: Review and discussion of the relevant background knowledge needed to complete the dissertation correctly.

- Chapter 3 – Design: Analysis of the tool's structure and its design.

- Chapter 4 – Implementation: The implementation as a tool in Python, along with the key methods and file structure.

- Chapter 5 – Evaluation: Review of the tool and any results which have been gathered, along with potential improvements or changes.

- Chapter 6 – Conclusions and Future Work: Overview of the dissertation and the conclusions drawn from it.

Chapter 2

Literature Review

2.1 Introduction

Information Extraction (IE) is the task of extracting pre-defined information from unstructured natural language text and presenting it in a structured format. The format structure varies by application, but typically resembles an OBJECT-VERB-OBJECT sequence. Once formatted, the information can be used in a range of different applications, such as in medicine, document curation, or simply used to populate a database [41] [47]. The desired information to be extracted is dependent on context, but typically includes entities, relations, and events, which generally follow a "who did what and where" structure [2]. IE systems typically focus on surface-level linguistic phenomena, such as parts of speech, morphological features, and syntactic dependencies to extract information [28].

IE systems operate on surface-level language-based phenomena present within text with the sole purpose of presenting information in a structured format. This differs from more complicated Text Understand (TU) systems which utilise more advanced techniques to decipher the subtle nuances present in text [2] [19]. For an information extraction system, the task is more limited than that of a text understand system, whereby only a small fraction of the text is relevant to the task and the nuances of meaning are not of particular interest. Conversely, text understanding systems aim to make sense of the text as a whole, whereby the full complexities of natural language are accommodated for [37].

The need for IE systems arose as a result of text processing within specialised domains, such as for the curation of biomedical articles, uses in government or military intelligence operations and advanced search algorithms for internet search engines [37] [47] [2]. The need for information extraction is highlighted in biomedicine in particular, where there are over 500,000 articles per year and large quantities of money are spent on their collective curation. Despite the varying contexts, most of the existing IE technologies were developed as features for systems which participated in conferences called the Message Understanding Conferences (MUC), which were held between 1987 and 1998 [2]. These conferences are explored in greater detail in Section 2.7.1.

All aspects of the Information Extraction task are language-agnostic, such that the process, output, and evaluation remain the same regardless of language. Due to this, the background research presented within this chapter applies to information extraction for any language, rather than just Swedish. A detailed and comprehensive evaluation of Swedish NLP resources, extraction techniques, etc. is presented within Chapter 3 where it is more relevant. Furthermore, a breakdown of the features and quirks of the Swedish language is presented at the end of this chapter.

2.2 Types of Information Structure Extracted

Within Information Extraction, the desired types of information structure for extraction are split into four categories. These four categories are: entities, the relationships which exist between entities, adjective-bound entities, and higher-order structures such as tables and lists [39].

2.2.1 Entities

Entities are typically noun phrases which comprise of one or more tokens. The most common form of entity is the Named Entity (NE), with the term first being coined in MUC-6, when Named Entity Recognition was first introduced [20]. NEs are split into various types, such as person names, locations, organisations, etc [6] [21] [39]. The classification of entity types is dependent on the application, with the Automatic Content Extraction (ACE) program

Type	Description
ORG	Companies, agencies, etc.
PRS	People, including fictional characters, etc.
GPE	Countries, cities, etc.
LOC	Non-GPE locations, such as lakes
DATE	Dates or time periods
MONEY	Monetary values, including the unit, etc.

Table 2.1: Some of the most common NE types

listing more than 100 different individual entity types, while modern systems are expanding definitions to include disease names, paper titles, etc [16] [39]. A breakdown of some of the most common entity types is listed in Table 2.1.

2.2.2 Relationships

Relationships follow an OBJECT-VERB-OBJECT structure, whereby the action performed by an entity is described. For example, "is acquired by" represents an association between two entities, whereby one entity is acquiring the other. Relationships can be either binary or multi-way, with multi-way relationship extraction being commonly referred to as record extraction [39].

2.3 Information Extraction Pipeline

The Information Extraction process can be divided into three distinct parts – local text analysis, discourse analysis and formatting. Within local text analysis, any individual pertinent pieces of information are extracted from the text [19]. This is achieved across multiple stages of the NLP pipeline, such as sentence segmentation and tokenisation, part-of-speech (POS) tagging, named entity recognition, partial syntactic analysis, and relation detection [39] [34]. Discourse analysis combines and integrates information to allow for a more detailed output in relation to the text. It comprises co-reference resolution and inferencing and event merging as stages. Finally, formatting is independent of the two aforementioned parts and ensures that extracted information is represented in the desired format [19].

2.3.1 Lexical Analysis (pre-processing)

Lexical analysis consists of sentence segmentation, tokenisation, and POS tagging. These techniques perform essential pre-processing on natural language text, ensuring that the later stages within local text analysis are possible. Sentence segmentation is the first stage of text pre-processing, whereby the boundaries of sentences are identified and an input text is split into its component sentences [19]. Following sentence segmentation, each sentence is decomposed into tokens. Tokenisation follows a pre-defined set of delimiters, such as spaces, commas, and full stops [39] [24]. Following tokenisation, each token is analysed to determine its part-of-speech and potential features through dictionary look-up techniques. During this stage, tokens are cross-referenced with general purpose dictionaries, as well as domain-specific dictionaries with domains such as major place names, common surnames, major company names, etc [19]. The set of assignable tags includes conventional parts of speech, such as nouns, verbs, adjectives, pronouns, etc, but is usually more detailed to capture a large amount of existing sub-types of the basic types.[39] [34].

2.3.2 Named Entity Recognition

Named Entity Recognition (NER) aims to identify named entities and other lexical structures present within natural language text [24]. NER systems identify different types of proper names, such as for persons and organisations, as well as lexical structures, such as dates and times [6]. These entities are identified using surface-level regular expressions which are defined to use parts-of-speech, syntactic features, and orthographic features [2] [19]. Named entities appear frequently within text, meaning that their identification and classification simplifies any processing completed further down the NLP pipeline. Furthermore, NEs represent important argument values for many information extraction tasks, therefore helping to identify the relationships which exist between entities [19]. An annotated example of Swedish NER is displayed in Figure 2.1.

$GCSE_{[qualification]}$:n var dock inte tillgängliga, och $Dan_{[name]}$ var tvungen att nöja sig med Law. $Tio\ år\ senare_{[time]}$ läser $Dan_{[name]}$ sitt sista år av $Computer\ Science\ BSc_{[qualification]}$ vid $Bangor\ University_{[school]}$ i $Wales_{[country]}$.

Figure 2.1: Swedish text with highlighted Named Entities

2.3.3 Partial Syntactic Analysis

Partial Syntactic Analysis is the process by which tokens are grouped into prominent phrase types through a process called chunking [39] [6]. Example phrase groups include verb phrases, noun phrases, and prepositional phrases. These groups are identified through a combination of syntactic and semantic evidence found within natural language text, typically through the use of a context free grammar [19] [39]. The features of each constituent token within a group can be cross-examined in later stages of the IE process. For example, verb group features may include tense, root form, and voice (whether the verb is passive or active).

2.3.4 Relation Detection

The final stage within local text analysis is relation detection, which is a form of dependency analysis. Relation detection focuses on identifying the likely relationships which exist between different entities within a text [6] [39]. The NER stage ensures that named entities are highlighted, allowing for the relationships between them to be extracted. These relationships are identified using similar regular expression-based techniques to that of NER, whereby POS tags and other syntactic features are used to isolate relationships [19].

2.3.5 Co-reference Resolution

Co-reference Resolution is the first task within the discourse analysis part of IE. The aim of this task is to eliminate and resolve any ambiguous references which are caused by pronouns or definite noun phrases within a text. This process usually relates any detected entities to any latter references which make use of an ambiguous pronoun [13].

2.3.6 Inferencing and Event Merging

Inferencing and Event Merging is the last stage of the IE process before the output is formatted. This stage aims to condense any disjointed partial information which may be spread across several sentences. Furthermore, any implied information which is not directly expressed needs to be inferenced such that it is defined explicitly within the output. This inferencing utilises handcrafted rules which are domain specific, thus providing the most accurate output [19].

2.4 Extraction Techniques

Typically, there are two main methodologies for the Information Extraction task – empirical and statistical. Empirical methods require the manual creation of linguistic rules or regular expressions for performing the extraction, while statistical methods make use of learning-based techniques to perform extraction [39]. The method utilised for a particular IE depends on the nature of the extraction task and the amount of noise present in the unstructured data, though some systems utilise both methodologies for hybrid systems [27]. Generally, rule-based empirical systems are easier to understand and develop, while statistical methods are more robust to noise within unstructured data. As such, rule-based systems perform best in closed-domain environments where human involvement is available, while statistical systems perform best for automatic extraction domains [39].

2.4.1 Empirical Approach

Many extraction tasks can be conveniently handled through a collection of rules. Early information extraction systems were all rule-based, such as the Alembic Information Extraction system, SRI FASTUS, and others [39] [24] [1]. Rule-based systems are faster and easier to optimise than machine learning-based systems, and perform best in controlled situations [24]. For example, when extracting phone numbers or email addresses.

Rule-based approaches make use of various features attached to each token within a text, such as [24]:

- The string representation of the token.
- Writing conventions of the token (e.g., capitalisation, symbols, etc).
- The POS of the token.
- Any annotations attached at earlier processing stages.

Rule-based approaches can also be employed for the assignment of POS tags within the POS tagging stage, but are generally less versatile than statistical methods. Difficulties can arise for ambiguous or unknown words, as the system relies on manually annotated rules, whereas a learning-based system could adapt and accommodate for future occurrences [24]. As such, statistical approaches are employed by two major NLP software libraries, the Natural Language Toolkit (NLTK) and spaCy. The NLTK uses a Hidden Markov Model for its POS tagger, with spaCy utilising statistical neural models [6] [46]. These software packages/libraries are explored further in Chapter 3.

2.4.2 Statistical Approach

Statistical NLP methods are derived from machine learning, which is a scientific method of learning-based data analysis [24]. Machine learning techniques can be categorised into two types: supervised and unsupervised. Both learning models are employed to construct probabilistic models of data [29]. The aim of supervised learning is to employ statistical methods to construct a prediction rule from labelled training data. Once trained, a supervised model will effectively imitate the training data in order to predict missing information in unlabelled texts [24]. Conversely, unsupervised systems are not trained using manually labelled data and aim to group data into clusters. These systems are more complex to implement, but alleviate the dependency on the creation of complex training datasets [4]. The composition and creation of labelled datasets is explored further in Section 2.5.1.

In statistical language modelling, large bodies of text are used to determine the parameters which are to be used within a language model [38]. Various language models exist, with examples including the Hidden Markov Model (HMM), Conditional Random Fields (CRF) and the Naive Bayes Model.

Language Models

Natural language can be perceived as a stochastic process, whereby bodies of text have a random probability distribution [38]. Furthermore, each sentence of n words within a text can be regarded as a sequence of n observations $\{x1, x2, ..., xn\}$ [24]. Language models aim to determine and utilise any regularities which exist within natural language to assign a probability to a sequence of words [38].

2.5 Statistical Model Training

As described in Section 2.4, various NLP models can be implemented as either a rule-based system or a learning-based system. In order to develop a sufficient learning-based system, large amounts of data are required to "train" the model. Depending on the type of learning, e.g., supervised, unsupervised, etc., the data changes slightly, but its role remains the same. All learning-based systems are powered by statistical models which perform decisions based on probabilities [3]. Training a model determines the best probabilistic weight values, as well as potential biases. Training is an iterative process where the model's predictions are compared against the training data in order to estimate the gradient of the loss. The model which minimises the loss makes the closest predictions to the training data and performs the most accurate annotations. Essentially, the accuracy of a learning-based NLP system is dependent on the size and quality of the training data.

The training data used for the development of learning-based NLP systems are usually presented in the form of a corpus. Typically, a corpus is split into training and testing sets, therefore allowing for a comparative evaluation. There is no optimal ratio for the splitting of data, but systems which are trained on a larger data set are more representative.

2.5.1 Training data - Corpora

A corpus is a collection of machine-readable authentic texts, which is sampled to be representative of a particular natural language. Corpora are essential for linguistic research, as well as the development of NLP systems, with a symbiotic relationship existing between the two fields. As research progresses within one field, the other benefits, and vice versa. Furthermore, computational linguistics has resulted in an increased level of corpus annotation, allowing for NLP tasks such as Part-of-Speech tagging, syntactic parsing, semantic tagging, and for the alignment of parallel corpora [24].

Various criteria are considered when a corpus is designed, such as its size, representativeness, balance, sampling quality, data capture and copyright, mark-up, and annotation. For NLP systems, using a larger corpus yields more reliable statistical modelling. Furthermore, a larger corpus provides a more accurate probability for the frequency of the distribution of a certain token in a text, hence producing better probability-based language models [3]. Corpora differ from archives or random text collections because of their representativeness. The representativeness is dependent on the criteria used for sampling.

Corpus annotation is the process by which raw natural language texts are encoded to allow for the extraction of linguistic information. Annotation is completed across different levels, with NLP being concerned with the lexical, syntactic, and discoursal levels. Lexical annotations include POS tags, lemmas for lemmatisation, and semantic fields for semantic annotation. The annotated tokens are represented according to various markup schemes, such as the Corpus Encoding Standard (CES). CES specifies a minimal encoding level that corpora must achieve to be considered as standardised. Additionally, CES is designed specifically for the encoding of language corpora and adopts both the Standard Generalised Markup Language (SGML) and the Extensible Markup Language (XML) for the representation of annotations [24].

An example of an annotated corpus is shown in Figure 2.2, which demonstrates the XML format used to represent natural language text. The annotations are simplified significantly for legibility, but include the lemma and Part-of-Speech. The sentence used translates to: "In the simple analysis, Iraq does not need missiles."

```
<corpus id="corpusID">
    <text id="textID">
        <sentence id="sentenceID">
            <w lemma="|i|" pos="PP">I</w>
            <w lemma="|den|" pos="DT">den</w>
            <w lemma="|enkel|" pos="JJ">enkla</w>
            <w lemma="|analys|" pos="NN">analysen</w>
            <w lemma="|behöva|" pos="VB">behöver</w>
            <w lemma="|inte|" pos="AB">inte</w>
            <ne name="Irak" type="LOC">
                <name type="inst">
                    <w lemma="|Irak|" pos="PM">Irak</w>
                </name>
            </ne>
            <w lemma="|missil|" pos="NN">missiler</w>
            <w lemma="|.|" pos="MAD">.</w>
        </sentence>
        ...
    </text>
    ...
</corpus>
```

Figure 2.2: XML sample taken from SUC 3.0 (see Chapter 3)

2.6 Output and Formatting

There are two primary aims of deployment for an Information Extraction system. The first aim is to identify all mentions of specified information within unstructured text, while the second aim is to populate a database of structured entities [39]. The format of the output is only relevant for the first aim, and once structured is referred to as an output template. The output template outlines the event and extracted information in a structured and easy-to-read format [10]. The extracted information regarding a takeover would be displayed as an output template similar to as in Figure 2.3.

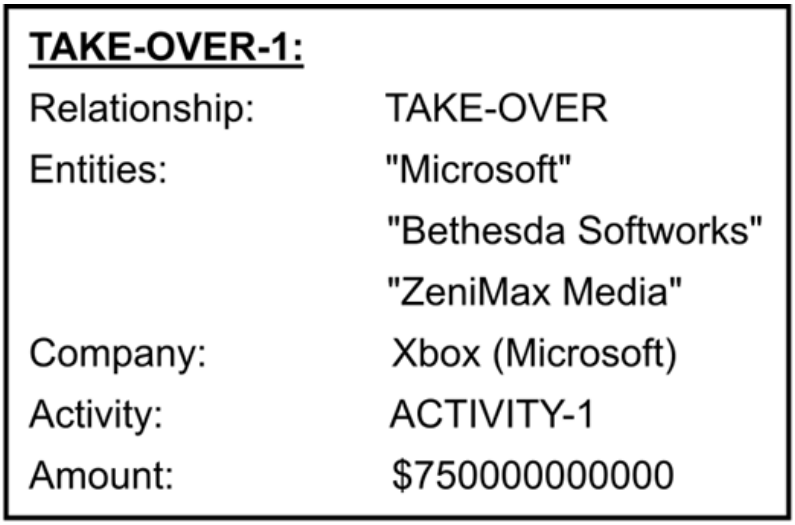

Figure 2.3: Example output template for acquisition sentence

2.7 Evaluation Metrics

Originally, there were no standardised evaluation metrics for information extraction. Following the first Message Understanding Conference (MUC), a set of evaluation metrics was required to differentiate between the participant systems. Starting with MUC-2, new primary evaluation measures were introduced – recall and precision [20]. Recall is a measure of how comprehensive the system is in its extraction of relevant information, while precision is a measure of the system's accuracy [1]. These evaluation metrics measure two important aspects of performance, as well as providing an overall view of the system performance. Additionally, recall and precision can be combined into a single score, known as the F-Measure. The F-Measure falls between recall and precision, whereby relative weights can be adjusted for different application requirements [11].

Recall is calculated as the number of correct answers divided by the number of possible correct answers [1]. The formula for calculating recall is:

$$recall = \frac{N_{correct}}{N_{possible}}$$

Precision is calculated as the number of correct answers divided by the number of answers provided [1]. The formula for calculating precision is:

$$precision = \frac{N_{correct}}{N_{provided}}$$

The formula for calculating F-Measure is:

$$F = \frac{(\beta^2 + 1) \times P \times R}{(\beta^2 \times P) + R}$$

where R is recall, P is precision, and β is the relative importance given to recall over precision. F-Measure is designed to deliberately yield a higher score for systems which have similar recall and precision scores, rather than vastly different [11]. If $\beta = 1$, then recall and precision are weighted equally. If $\beta < 1$, recall is more significant, otherwise, precision is more significant [23].

2.7.1 Message Understanding Conferences

The Message Understanding Conferences (MUCs) were initiated by the Naval Ocean Systems Centre (NOSC) with the support of the Defence Advanced Research Projects Agency (DARPA) to assess and perform research on methods for executing automated analysis of text-based military messages. Each conference required a participating group to develop a system capable of extracting desired information from provided sample text messages. These sample messages and instructions would be provided prior to each conference, and reasonable success in the task was required in order to attend the conference. To evaluate the effectiveness of each system, the participating groups would be provided test messages to be run through each system shortly before the start of the conference, with the outputs being evaluated against a manually prepared answer set [20].

The MUCs are notable because they significantly helped to shape the research program in information extraction and brought the field to its current state [20]. Prior to the MUCs, most existing systems relied on heuristics and handcrafted rules similar to a symbolic system. Progression within the field

stopped and the number of new publications for the field stagnated. Following the creation of MUC, further approaches that were statistical in nature were presented and the number of new publications for the field increased with MUC-6 in 1996 [30]. In addition to reigniting research within the field, the MUCs also established definitions and terminologies as a standard to be used with reference to information extraction. Furthermore, because the MUCs were evaluation-based, primary evaluation measures were developed to ensure systems met the requirements of the brief. These evaluation measures are known as precision and recall, from which an F-Measure can also be calculated [20].

Following MUC-6, various other conferences and events were organised which aimed to improve information extraction as a field, with examples like The Information Retrieval and Extraction Exercise (IREX) in 2000 and The Conference on Computational Natural Language Learning (CoNLL) in the years 2002 and 2003. IREX presented two tasks which could be participated in: an information retrieval task and a named entity recognition task. CoNLL aimed to pit differing attempts at language independent NER systems against each other, with a concentration on persons, locations, organisations and miscellaneous as the named entities. The corpora used during the 2002 conference were in both Dutch and Spanish, while the 2003 conference used English and German corpora [43] [44].

2.7.2 SRI FASTUS

Developed in December 1991, the Finite State Automata-based Text Understanding System (FASTUS) is an information retrieval system created by SRI International [23]. FASTUS was designed to address the need for an accurate high-speed solution for extracting pre-specified information from within a text. The system operates using a cascading non-deterministic finite state language model which produces a phrasal decomposition of a sentence into noun groups, verb groups, prepositions, conjunctions, and relative pronouns through syntactic parsing techniques [1]. FASTUS was originally designed as a pre-processor for an existing text-understanding

system called TACITUS, but was found to be considerably faster. FASTUS could process 100 messages in under 12 minutes, while TACITUS required 36 hours for the same set of messages. This improved performance led to FASTUS succeeding its precursor and allowed for faster development of the system, and for its participation in various Message Understanding Conferences (MUC) [23].

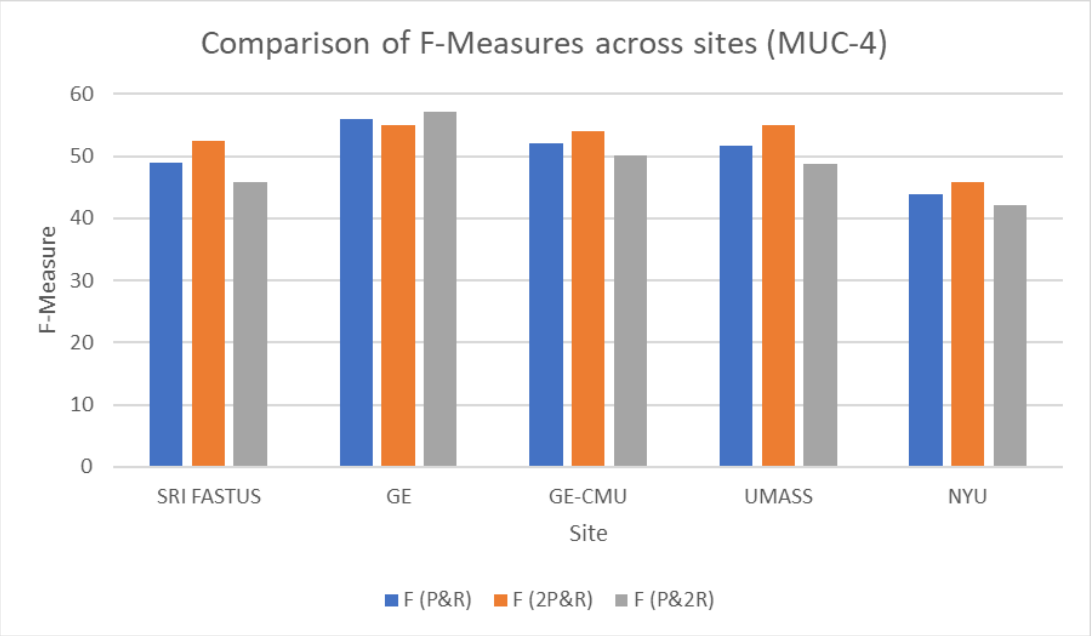

Figure 2.4: F-Measures across some of the participating MUC-4 sites

The effectiveness of the FASTUS system has been evaluated for each MUC which it participated in. For example, the MUC-4 evaluation of 100 unseen texts yielded a recall of 44%, a precision of 55% and an F-Measure of 48.9. A second test using texts from a different time span to the provided training messages yielded a recall of 52%, a precision of 44% and an F-Measure of 47.7. These scores were the second highest of the 17 sites tested, with FASTUS being able to read 2,375 words per minute or one text every 9.6 seconds on average, making it an order of magnitude faster than the other leading MUC-4 systems. In addition, the MUC-5 version of FASTUS was capable of handling both English and Japanese in the information extraction task. The English task yielded a recall of 34%, a precision of 56% and an F-Measure of 42.67, while the Japanese task yielded a recall of 34%, a precision of 62% and an F-Measure of 44.21 [23].

2.8 Swedish

Swedish is a fully developed Germanic language which is spoken by approximately 13 million people internationally, with 9.4 million native speakers [17]. Linguistically, Swedish has moderately complex inflectional morphology [33], and features two grammatical genders, common and neuter [40]. Swedish follows the V2-rule for word order, whereby the finite verb is always situated in the second grammatical position of a declarative clause. This is dissimilar from English, which features an exclusively Subject-Verb-Object (SVO) word order. For sentences which begin with the subject, there is similarity in word order, as shown in Table 2.2.

Subject	**Verb**	**Object**	**Translation**
Jag	äter	ris	I eat rice
Han	älskar	henne	He loves her
Jag	gillar	det	I like that

Table 2.2: Simple SVO Swedish sentence comparison

However, the word order changes in sentences which do not begin with the subject:

Imorgon åker jag till Sverige (OBJECT VERB SUBJECT)
I'm going to Sweden tomorrow (SUBJECT VERB OBJECT)

Furthermore, sentences which begin with a subordinate clause highlight how grammatical position differs from literal position. Grammatical position refers to each individual element within a sentence, while literal position refers to the individual word. Therefore, a preceding subordinate clause may contain 4 words, but still be in position 1. For example:

När jag var ung ville jag ha ett Xbox (SUB-CLAUSE VERB SUBJECT OBJECT)
When I was young, I wanted an Xbox (SUB-CLAUSE SUBJECT VERB OBJECT)

These aspects are important to consider when designing a pattern-based information extraction system. As such, the various considerations are explored further in Chapters 3 and 4.

2.9 Summary and Discussion

This chapter explored the information extraction task as a whole, whereby various aspects, such as the IE pipeline, extraction techniques, etc. were explored in detail. The completion of this research ensures that the correct resources and methodologies are utilised in the tool's design (see Chapter 3).

Furthermore, this chapter also featured a brief overview of the Swedish language and its unique complexities. These complexities are important as they form the basis of the implementation and are utilised by the various technologies explored in relation to IE.

Chapter 3

Design

3.1 Software Breakdown

In order to create a novel tool for information extraction of Swedish text, an appropriate choice in programming language was required. No single programming language exceeds another with reference to NLP task capabilities, with many existing NLP systems being available across a range of different programming languages. For example, the Natural Language Toolkit (NLTK) which is available in Python [5], the Tawa toolkit which is available in C, and CoreNLP which is available in Java [42]. These systems are outlined further in Table 3.1.

After browsing the various existing NLP systems, it was decided that Python would be the best choice for the information extraction task. Being considerably more comfortable with Java, this was a difficult design decision, especially having made various Java-based applications in the past. However, Python features industry-leading NLP libraries and frameworks such as the NLTK [5] and spaCy [15], as well as pandas for data visualisation [45]. Furthermore, despite various Swedish resources for other programming languages, such as Stagger for Java [33], Python had the best Swedish support across all aspects of the NLP pipeline.

System	Type of System	Language
CoreNLP	suite of tools	Java
NLTK	library	Python
Tawa	toolkit, libraries	C

Table 3.1: Some NLP systems and their details [42]

3.1.1 Python for NLP

Python is an easy-to-learn programming language which features a transparent syntax and good string-handling functionality. Python is packaged with an extensive standard library, which includes graphical programming and numerical processing tools, making it particularly useful for a wide range of applications out of the box [5]. Furthermore, Python is used in industry, research, and education worldwide. It has garnered praise for the way in which it facilitates productivity and quality, as well as how it ensures software maintainability [6].

There are various well-documented, industry-leading NLP libraries and frameworks available with Python, such as the NLTK [5], spaCy [15], etc. These libraries also have numerous internet-based resources which can be used, such as tutorials and walkthroughs. These resources are especially important for languages which are not natively supported, such as Swedish.

3.1.2 The Natural Language Toolkit (NLTK)

The Natural Language Toolkit (NLTK) is an essential framework which is widely used for teaching and research into NLP. Developed by Steven Bird and Edward Loper, the library played a key role in the breakthrough in NLP research [5]. Various NLP tasks are supported by NLTK, including chunking, classification, stemming, parsing, lemmatisation, POS tagging, semantic reasoning and tokenisation, etc. An example regular expression-based chunking output is shown in Figure 3.1.

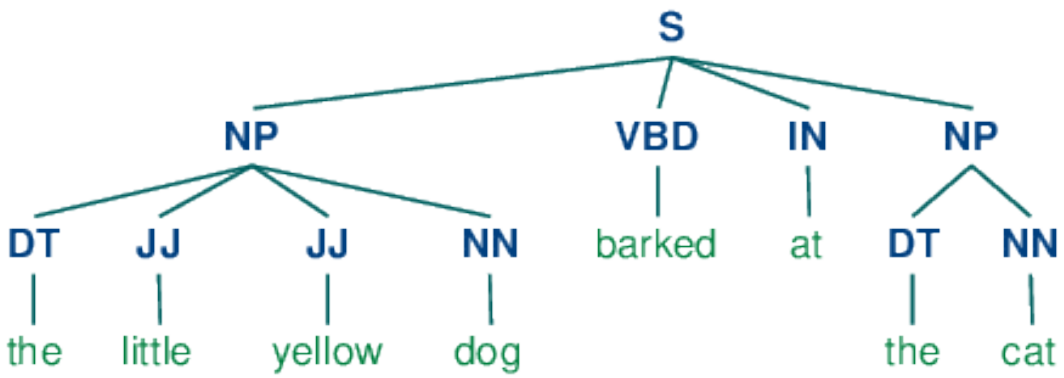

Figure 3.1: Example of a simple regular expression-based Noun Phrase Chunker [6]

3.1.3 spaCy

spaCy is an open-source NLP software library which is written in Python. Unlike NLTK, which is designed primarily for teaching and research purposes, spaCy is designed for production usage [15]. Various NLP tasks are supported by spaCy, including tokenisation, POS tagging, dependency parsing, NER, rule-based matching, etc. Furthermore, spaCy features a transition-based greedy parser which is currently the fastest syntactic parser available [12].

spaCy features convolutional neural network (CNN) models for its POS tagging, dependency parsing, text categorisation, and NER software [46]. Furthermore, pre-built statistical neural network models are available to perform these tasks across 17 natively supported languages, including English, Spanish, and Chinese, but unfortunately not Swedish.

3.1.4 pandas

pandas is a package built atop NumPy which offers a convenient storage interface for labelled data. Data visualisation is simple and easy through pandas, particularly for data science applications like information extraction. An example formatted output is in Figure 3.2. pandas also implements various useful data operations similar to those within database and spreadsheet programs [45].

ID	Sentence	Length	Descriptive noun
1	Om detta nödvändiga arbete däremot utförs i vä...	29	nödvändiga arbete
1	Om detta nödvändiga arbete däremot utförs i vä...	29	vettigt sätt
2	Vad är då kristen etik, eftersom, Alf Svensson...	23	kristen etik

Figure 3.2: Example pandas output for Swedish text

pandas features various efficient implementations and features, such as the DataFrame and Series objects. A DataFrame is a multidimensional array with attached column/row labels which allows for heterogeneous types and empty values. A Series is a one-dimensional array of indexed data and functions very similarly to a NumPy array. These data types are useful as they allow for the efficient access, comparison, and output of stored data.

3.2 Swedish NLP

3.2.1 Språkbanken

Språkbanken (The Swedish Language Bank) is a research unit established in 1975. Originally designed to collect, process and store Swedish text corpora and other linguistic resources, Språkbanken has since expanded its focus to their development instead. Various tools are available to use through Språkbanken, including the corpus tool, Korp. The Korp corpus pipeline is used for importing, annotating, and exporting corpora into different formats [8]. Annotations are completed automatically by the corpus annotation pipeline infrastructure, Sparv. Currently, Sparv provides the following annotations: tokenisation, sentence segmentation, lemmatisation, etc [7]. Sparv reduces the amount of time and money required for the annotation of texts; however, because the process is automatic rather than manual, its annotations cannot be considered as a gold standard. In addition to the various tools and lexicons, Språkbanken also hosts various pre-trained dependency parsing models, lemmatisation models, and POS taggers. Some of the hosted corpora are listed as being training and evaluation data, including the Universal Dependencies treebank Talbanken, as well as versions 2 and 3 of the Stockholm-Umeå Corpus (SUC). These corpora are explored and compared in Section 3.5.

3.2.2 Pre-trained POS Taggers

There are various Swedish POS taggers available, such as Stagger, Flair, and MarMoT. Stagger is the most accurate single tagger for Swedish, reaching an accuracy of 96.4% when evaluated against the Stockholm-Umeå Corpus (SUC). In addition, Stagger has also been trained on internet-based sources which do not adhere to traditional language norms, such as the Stockholm Internet Corpus (SIC) [33]. Currently, Stagger is available as source code or a JAR executable.

3.2.3 Pre-trained Dependency Parsers

Stanza is an open-source natural language toolkit which supports 66 languages. Currently, Stanza features a language-independent neural pipeline which performs tokenisation, lemmatisation, POS and morphological feature tagging, dependency parsing, and NER. The software was trained on 112 datasets, including the Universal Dependencies treebanks [36]. As such, Stanza would have been trained with the Talbanken corpus, which is explored in Section 3.5. Stanza is currently available in Python, but features a native interface to the Java CoreNLP software, which further extends its functionality.

3.2.4 Pre-trained spaCy Pipeline

spaCy currently supports 16 languages, not including Swedish. Due to the lack of language support or language-agnostic functionality, various packages and libraries which extend spaCy functionality have been developed. Some examples include the spaCy + UDPipe and Swedish spaCy packages.

The spaCy + UDPipe package is inspired by Stanza, offering slightly less accurate models which are subsequently much faster. The package wraps the UDPipe language-agnostic NLP pipeline such that it can be used in conjunction with spaCy. Unlike the Swedish spaCy package, this model is not designed as a Swedish pipeline, instead aiming to be language independent.

Swedish spaCy includes a POS tagger, dependency parser, and NER. Two separate models were trained in consideration of Universal POS tags (UPOS) and Language Specific POS tags (XPOS). In addition to the language models, complete spaCy pipelines have also been developed, including a POS tagger, dependency parser, sentencer, NER, and lemmatiser. The UPOS tagging model was trained using the Universal Dependencies Swedish-Talbanken treebank (see Section 3.5.2) and yielded an accuracy of 96.37, while the XPOS tagging model was trained using the SUC 3.0 corpus (see Section 3.5.1) and yielded an accuracy of 96.84. Furthermore, the NER was also trained using the SUC 3.0 corpus and yielded a recall of 84.48, precision of 86.27, and an F-Measure of 85.37.

3.3 Pipeline Breakdown

3.3.1 Custom POS Tagger

Stochastic POS taggers have considerably lower error rates than rule-based alternatives [9] and are generally more versatile in comparison, yet their development is complex and time-consuming. Rule-based taggers sacrifice accuracy for other benefits, such as a reduction in required stored information, increased ease of finding and implementing improvements, and better portability [9]. However, the reduced accuracy of a rule-based POS tagger produces a ripple effect, whereby latter stages of the NLP pipeline are affected – reducing overall system accuracy.

A comparison of the two approaches yields the conclusion that stochastic POS taggers are more accurate, but considerably more difficult to construct. Conversely, a rule-based tagger would be less accurate and also potentially difficult to construct, especially considering that a native Swedish speaker would be needed at the least. Linguists who understand the complex rules of Swedish morphology would be better suited to the task and are still outperformed by stochastic methods, so my attempts would likely be considerably worse. As such, it was decided that utilising a pre-made library package would provide the best results and not affect the accuracy of the entire pipeline.

As discussed in Section 3.2.2, there are various Swedish POS taggers which can be used. Stagger is the most accurate single tagger for Swedish, but predominantly written to be used in conjunction with CoreNLP through Java [33]. Stagger could be used with Python through the use of various libraries, but this practice seems fragile and like potential overkill. An alternative option is the POS tagger included in the Swedish spaCy package, which utilises a convolutional neural network. This tagger was trained on SUC 3.0, features a language specific tagset (XPOS), and has an accuracy of 96.84. This tagger performed well with various corpora, including SUC 3.0, Talbanken, and 8-SIDOR, and is therefore used in the custom pipeline.

3.3.2 Custom NER

Rule-based and statistical approaches to NER often yield similar results, regardless of context [18]. The choice of approach for a custom pipeline depends on the complexity of the NER, i.e., the amount of tags, the complexity of the tags, etc. For instance, recognising names would be relatively simple with a rule-based approach. Annotated POS tags would highlight proper nouns with a high level of accuracy, and custom grammar rules could identify particular morphological features which indicate a name.

Seeing as the aim of the dissertation is to design a versatile tool for Swedish information extraction, it was determined best to implement a hybrid NER system. This hybrid system utilises the Swedish spaCy NER module and a custom rule-based approach. The swedish spaCy NER was trained on SUC 3.0 and yielded a recall of 84.48, a precision of 86.27, and an F-Measure of 85.37. The custom rule-set operates by identifying proper nouns and potential NERs which are incorrectly tagged. For instance, "ZeniMax Media" is incorrectly identified as "ZeniMax" by the spaCy NER because "Media" is tagged as a noun, rather than a proper noun. The custom rule-set catches this incorrect tag and identifies the full entity as a company name. The various custom NER rules and operations are outlined fully in Chapter 4.

3.3.3 Custom Phrase Detection

For phrase detection, it was decided that manually analysing the morphological features and dependencies of Swedish text would allow for the creation of a custom rule-set. To better analyse and understand Swedish morphological features, a parser module was developed. This module displayed morphological features and made use of the dependency visualiser, displaCy, to analyse dependencies.

Using the custom rule-set, extracted phrases include: descriptive noun phrases, preposition noun phrases, and verb phrases. Any extracted phrases are output in a formatted and labelled table through pandas, as well as the percentage of sentences which matched the phrase pattern. The noun

phrases can be used in conjunction with the verb phrases to build up a detailed understanding of the information present within a sentence.

3.4 Syntax and Typed Dependencies

When developing a rule-based NLP system, it is imperative for the programmer to appreciate and understand the subtle complexities of natural language. These complexities typically arise with regard to syntax and are significant as the veracity of the defined rules is dependent on the programmer's understanding of the language [39]. This is especially significant for systems which perform NLP tasks on languages which aren't native to the programmer. To combat these restrictions, various representations have been developed, such as the Universal Dependencies (UD) framework.

Universal Dependencies (UD) is a framework which provides consistent annotation of grammar across a variety of different languages. Annotated grammar types include parts-of-speech, morphological features, and syntactic dependencies. The aim of UD is to influence multilingual parser development, parser research with reference to language topology, and cross-lingual learning. The goal of this aim is to ensure the consistent annotation of similar syntactic structures across all languages, while still allowing for language-specific extensions [32]. The UD scheme is based on a combination of the universal Stanford Dependencies [14] and the Google universal part-of-speech tags [35].

For Swedish, all 17 universal UD POS tags are utilised. The full tagset is outlined in Table 3.2. These tags are important for recognising NEs and defining patterns. A more detailed explanation of how POS tags are used is available in Chapter 4, with an overview of Swedish-corpora specific POS tags in Section 3.5.

Additionally, Swedish features 9 dependency sub-types on top of the standard universal set. The standard UD set contains many different dependency types, such as nsubj, csubj, etc. The Swedish-specific additional dependencies are

Code	Category
ADJ	Adjective
ADV	Adverb
INTJ	Interjection
NOUN	Noun
PROPN	Proper Noun
VERB	Verb
ADP	Adposition
AUX	Auxiliary
CCONJ	Coordinating Conjunction
DET	Determiner
NUM	Numeral
PART	Particle
PRON	Pronoun
SCONJ	Subordinating Conjunction
PUNCT	Punctuation
SYM	Symbol
X	Other

Table 3.2: The 17 Universal UD POS Tags [32]

outlined in Table 3.3. These dependencies are important for identifying and extracting phrases, with a more detailed run-down in Chapter 4.

Code	Meaning
acl:cleft	Dependent clause in cleft sentence
acl:relcl	Relative clause modifier
aux:pass	Passive auxiliary
compound:prt	Verb particle
cubj:pass	Clausal passive subject
flat:name	Complex names with no clear head
nmod:poss	Possessive nominal modifier
nsubj:pass	Passive nominal subject
obl:agent	Agent modifier for passive verbs

Table 3.3: The 9 additional Swedish dependency sub-types

3.5 Corpora

There are numerous unlabelled corpora available for Swedish, but a limited amount of labelled data which can be used for training/testing of an NLP model. Of the 19 hosted corpora on Språkbanken, most are either not substantial enough, outdated, or too heavily specialised. The corpora which would be applicable to the design context are outlined in Table 3.4.

Name	Number of Tokens	Number of Sentences
SUC 3.0	1,166,593	74,245
SIC2	13,562	892
Swedish-Talbanken	96,346	6,160

Table 3.4: Comparison of various Swedish corpora

The Swedish Internet Corpus (SIC2) consists of Swedish blog posts which are annotated with POS tags, morphological features, and named entities. This corpus is considerably smaller than SUC 3.0 and Swedish-Talbanken, and more specialised to a certain context. The other corpora are investigated in greater detail below.

3.5.1 Stockholm-Umeå Corpus

The Stockholm-Umeå Corpus (SUC) is a balanced corpus of written Swedish which contains over 1 million tokens. Each token is annotated with its lemma, POS, and any named entity information. Work began on the corpus in 1989, and it was compiled in the 1990s at Stockholm University and the University of Umeå. There have been three versions of SUC, with version 1.0 being released in 1997, version 2.0 in 2006 and version 3.0 in 2012. Since its release, SUC has been the de-facto standard for tagging Swedish text [33]. Two variants of SUC 3.0 are available for use – the official corpus, and the scrambled corpus. The official corpus features manual annotations and conforms to a gold standard. Conversely, the scrambled corpus has its sentences scrambled and also includes additional automatic annotation by the corpus annotation pipeline infrastructure, Sparv. Due to these additional annotations not being completed manually, this version of the corpus is not a gold standard.

SUC consists of 500 text samples, each with a length of 2000 words. SUC 3.0 develops this further by dividing the samples into a training-development-test split [33]. Of the 500 samples, 373 are informative prose pieces and the remaining 127 are imaginative prose pieces. Informative Prose is split across 9 unique categories, each with differing distributions, while Imaginative Prose is split across 4 unique categories, also with differing distributions [22]. The categories are outlined in Table 3.5. Note that category D, Religion, has been spread between categories E, F, and J.

Category Symbol	Category Title	Distribution
I.	Informative Prose	**373**
A.	Press: Reportage	44
B.	Press: Editorial	17
C.	Press: Reviews	27
D.	Religion	0
E.	Skills, Trades and Hobbies	58
F.	Popular Lore	48
G.	Belles-Lettres, Biography & Memoirs	26
H.	Miscellaneous	70
J.	Learned and Scientific Writing	83
II.	Imaginative Prose	**127**
KK.	General Fiction	82
KL.	Mysteries (L) and Science Fiction (M)	19
KN.	Light Reading (N + P)	20
KR.	Humour	6

Table 3.5: SUC 3.0 categories and token distribution [22].

Category	Tag	Description
Person	PRS	People names (forenames, surnames), etc.
Location	LOC	Functional, geographical, geo-political, etc.
Organisation	ORG	Political, athletic, military, media, etc.
Artifact	OBJ	Food/wine products, prizes, vehicles, etc.
Work & Art	WRK	Printed material, names of films, etc.
Event	EVN	Religious, athletic, scientific, battles, etc.
Measure	MSR	Volume, age, index, dosage, etc.
Temporal	TME	Time values, etc.

Table 3.6: SUC 3.0 NER tagset [26].

As explored in Section 2.5.1, corpora feature a large proportion of sentences which are comprised of annotated tokens. Named Entities (NEs) are also annotated within corpora, but are additionally labelled with a tag which represents their category. The SUC 3.0 corpus features a relatively short tagset which is shown in Table 3.6.

3.5.2 Swedish-Talbanken Treebank

The Swedish-Talbanken treebank is a corpus based on Talbanken, a corpus created in 1974 at Lund University. The original Talbanken treebank was a syntactically annotated corpus of written and spoken Swedish which contained around 320,000 tokens. The original treebank was constructed before the switch to digital information-storing, such that it was placed on punch cards

and utilised an outdated format, MAMBA [31]. As such, conversions were necessary for the modern Swedish-Talbanken implementation.

Of the original four parts, only the first (professional prose) was converted for the Swedish-Talbanken treebank. It consists of roughly 6,000 sentences and 96,000 tokens which have been taken from a variety of informative text genres, including articles, brochures, and textbooks. Each token is annotated with its lemma, POS, features, and relations. These annotations were all originally manual but were automatically converted to the Universal Dependencies (UD) representation. Some of these automatic annotations were subsequently manually corrected.

Talbanken utilises 16 of the possible 17 UD tags as shown in Table 3.2, with the exception of the POS tag “X”. Furthermore, all of the additional dependency sub-types as listed in Table 3.3 are utilised by Talbanken.

3.6 Summary and Discussion

This chapter built upon the research performed in Chapter 2, whereby the relevant and specific aspects of the IE task are applied to the context of the application – the Swedish language. Various Swedish NLP resources, such as POS taggers, corpora, and existing libraries have been explored and compared. Furthermore, various design decisions, such as which libraries and corpora to use have been outlined and justified.

The most important design decisions are summarised below:

- The use of Python as the programming language for the tool.
- The choice of the Swedish spaCy package over Stagger, etc.
- The design decision for a custom hybrid NER approach.
- The use of SUC 3.0 as the training and testing corpus.

Chapter 4

Implementation

4.1 Introduction

This chapter discusses the implementation of a novel tool for information extraction of Swedish text. Each section provides an explanation of a specific part of the system, whereby every aspect of the system is described along with the implementation. Where applicable, screenshots from the command line interface and pseudo-code are provided.

4.2 Software Stack

4.2.1 Required Libraries

The libraries which were used to create this tool are as follows:

- NLTK - NLP package. Used for various NLP tools, such as tokenisation. Preferred version: 3.5.
- spaCy - NLP package. Used for model training, NER, etc. Preferred version: 2.3.5.
- pandas - Data analysis package. Used for visualising data, such as relationships. Preferred version: 1.2.3.
- lemmy - Lemmatiser package. Used for lemmatising tokens. Preferred version: 2.1.0.

In addition to imported packages and libraries, this tool also makes use of sb_corpus_reader.py, which was written to work with Språkbanken corpora, and sv_model_xpos, a pre-trained NLP model. Both of these resources are explored in later sections.

4.2.2 Files and Folder Structure

The tool is split across multiple files into four main components: the command line interface, the parser module, the NER module, and the information extraction module. Each of these modules and their functionalities will be explored further in this chapter. A directory tree displaying the folder structure is shown in Figure 4.1.

input_data

This folder is the location in which any input data are to be stored. Input data should be stored with the .txt extension, whereby information is represented line by line. The tool operates by reading each line of the file as an input doc, therefore a four sentence paragraph should be represented on a single line. Furthermore, it is important that input files utilise UTF-8 encoding, otherwise the various additional Swedish characters will be improperly represented.

This folder contains two text files, "empty.txt" and "sample_swedish.txt". "empty.txt" is an empty file used to demonstrate error handling, while "sample_swedish.txt" contains a few handwritten test docs which each consist of a few sentences. These docs were written predominantly for the NER module, but can be applied to both the parser and information extraction modules, too.

models

This folder is the location in which any Swedish language models are to be stored. Language models are typically trained on context-specific corpora, therefore having multiple different usable models yields a broader use-case for the tool. Out of the box, one model is provided, "sv_model_xpos". This

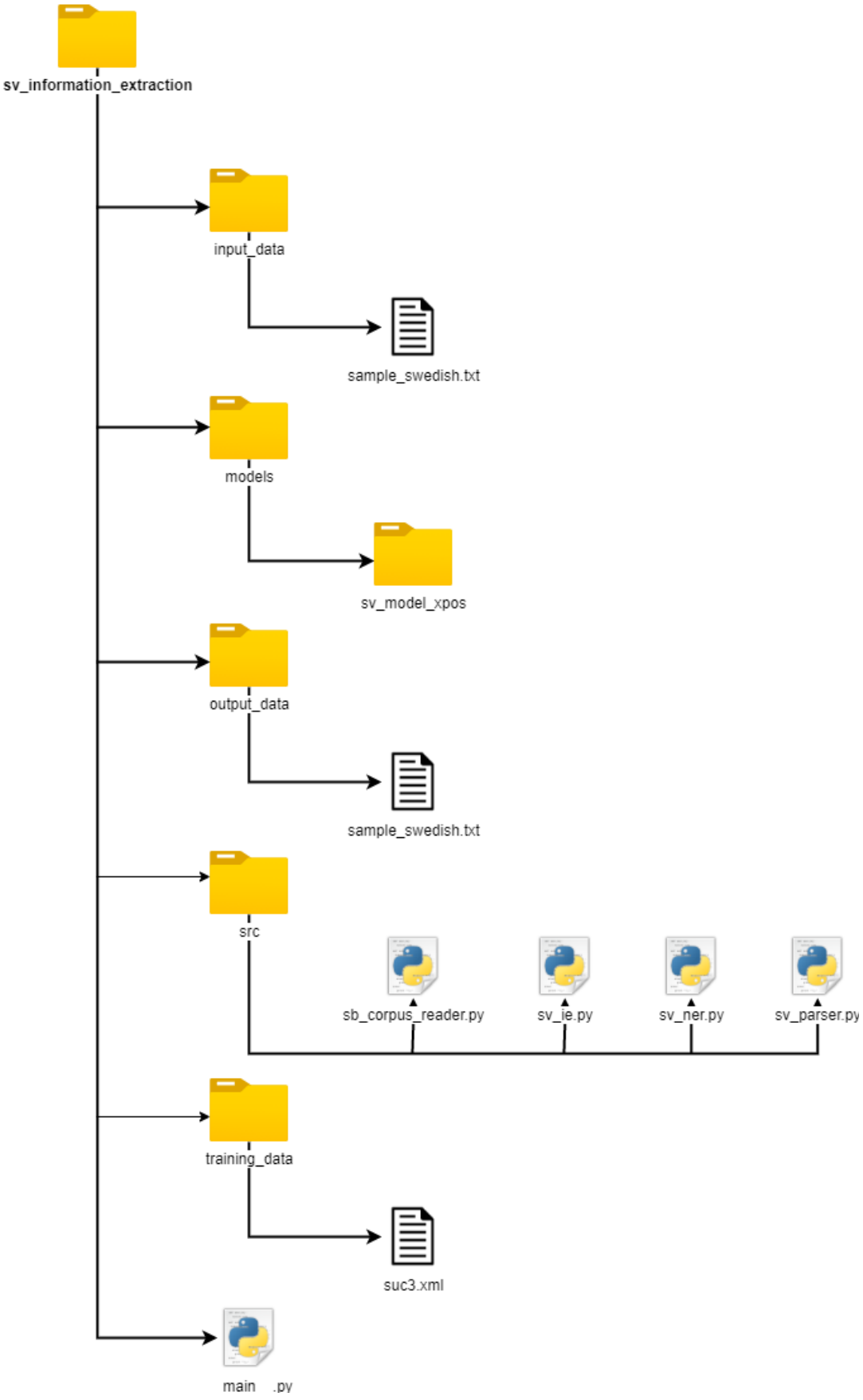

Figure 4.1: Folder structure for the Swedish Information Extraction Tool

model was trained on the SUC 3.0 corpus (see sections 2 and 3). The selection of a model is completed via the command-line.

output_data

This folder contains a single text file, "sample_swedish.txt", which is used for storing any docs which are saved via the command-line. These docs are represented in the same format as in the "input_data" folder, thus allowing for re-usability. Docs are separated by a newline for enhanced readability.

src

This folder contains four source code files: "sb_corpus_reader.py", "sv_parser.py", "sv_ner.py", and "sv_ie.py". "sb_corpus_reader.py" is a specialised corpus reader which was designed to read Språkbanken corpora into the NLTK. The other three aforementioned files contain the parser, NER, and IE modules, respectively. These are explored further in the sections below.

training_data

This folder is the location in which the SUC 3.0 corpus (see Section 3.5) is stored for demonstration and testing purposes. Furthermore, as explored in Chapter 6, this corpus may also be used for training further Swedish language models.

__main__.py

The "__main__.py" module contains the main command-line interface and is the centre of the tool's functionality. The Python interpreter automatically runs this file on the command-line, easing usability for the end-user. All of the other modules (see above) are accessed through this module.

4.2.3 Implementation Difficulties

As explored in chapters 2 and 3, there is a considerable lack of Swedish NLP resources available. Even veteran NLP packages such as NLTK [6] don't natively support Swedish, thus making it difficult to develop an Information

Extraction tool. This is particularly significant for aspects of the pipeline such as the POS tagger, whereby a professional Swedish linguist who understands the syntactic and morphological structure of the language is needed to produce an accurate system.

Additionally, the training and testing of an NLP model utilises considerable amounts of processing power. This is problematic as the development hardware used is not particularly powerful, therefore often resulting in slow loading and an increased development time. For example, loading a pre-trained Swedish model produces a noticeable run-time delay, as well as opening a moderately sized corpus, such as SUC 3.0 (see Chapter 3).

4.3 Interface Design

There were two main distribution directions available for this solution: a runnable command-line tool, or an importable package. An importable package represented a more complete release and appeared to be more professional, but it was felt that it did not represent the general-purpose, visualisation-centred approach of the software. An importable package would make more sense for a specialised application, such as for military [23] or medical [47] uses. As such, the runnable command-line tool was chosen as it allowed for greater data visualisation and promoted the contextually non-specialised nature of my information extraction implementation.

4.3.1 Using the Tool

Use of the tool requires a small amount of Python knowledge due to the lack of a window-based Graphical User Interface (GUI), but it is a relatively small learning curve. To help the user, instructions are provided within the "README.txt" file. Furthermore, a help message is displayed if the user enters incorrect command-line arguments.

To run the tool, the user needs to be in the "sv-information-extraction" directory, which can be navigated to via the command-line. From

there, the tool can be started using the command format "*python sv_information_extraction mode source*", whereby:

- mode can be either: "*parse*" for the parser module, "*ner*" for the ner module, or "*ie*" for the information extraction module

- source can be either a filename, such as "*sample_swedish.txt*", which is provided out of the box, or "*--sample*" which uses excerpts from SUC 3.0.

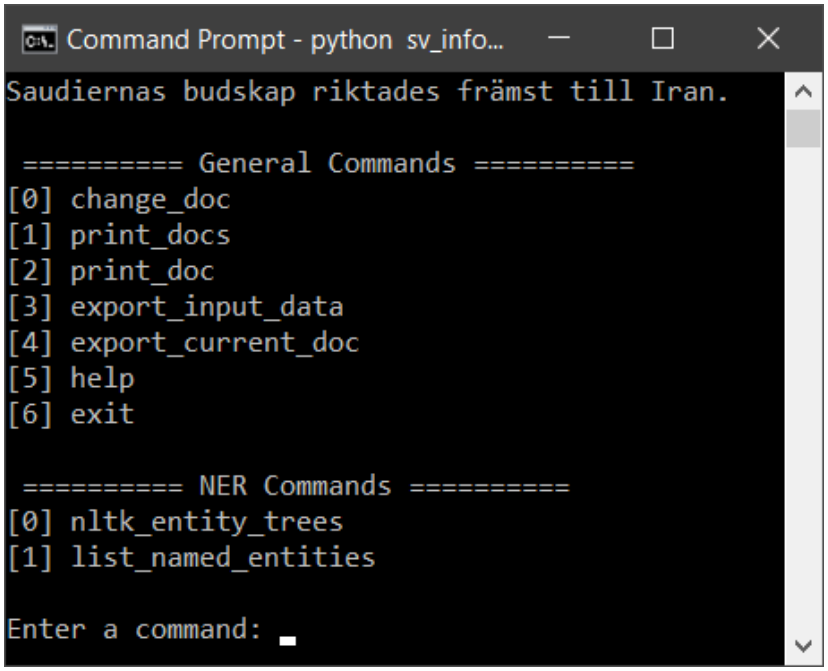

Figure 4.2: Information Extraction command-line interface

After the tool has been started, the user needs to select a language model and a doc to initially use. The available options for each are displayed and can be selected by entering their respective indices. Upon the completion of these stages, the user will be brought to the main menu. The main menu features two different mini-menus, the general commands menu, and the mode specific commands menu. The general commands are available across all selected modes and are shown in Figure 4.2. The mode specific commands are explored further in their respective sections below.

Method	Parameters	Description
split_sents	Plaintext	Splits an input doc into sentences
print_sents	Plaintext	Prints out each sentence in a doc
print_syntactic_info	spaCy doc, spaCy model	Prints out morphological features for each doc
remove_stopwords	Plaintext, spaCy model	Removes any stopwords from the input text
print_word_frequency_list	spaCy doc, spaCy model	Prints frequency of tokens in the input doc, not including stopwords
print_pos_frequency_list	spaCy doc	Prints frequency of POS tags in the input doc
print_tokens	spaCy doc, spaCy model	Prints out each token within a doc and the starting index
print_stopwords	spaCy doc	Prints out any stopwords in the input doc
print_dependency_skeleton	spaCy doc, spaCy model	Prints out any tokens and their dependencies
print_lemmatise_doc	spaCy doc, spaCy model	Prints out any tokens and their lemmas

Table 4.1: Overview of ie_parser.py functions

4.4 Parser

The parser module features 10 methods, 8 of which are accessible via the command-line interface. These methods are designed to highlight the morphological and syntactic features of the selected doc, with an emphasis on increased data visualisation and sentence structure analysis. The various methods and their descriptions are highlighted in Table 4.1.

4.4.1 syntactic_info

This method splits an input doc into sentences, with each sentence being split into its component tokens. Tokens are output with their syntactic dependency, simple POS tag, and complex POS tag in a tabular format, as shown in Figure 4.3. This process is relatively simple due to spaCy which assigns attributes, such as .dep__, .pos__, etc.

4.4.2 print_dependency_skeleton

This method highlights the syntactic dependencies which exist between tokens within a sentence. The output is of a tabular format and includes

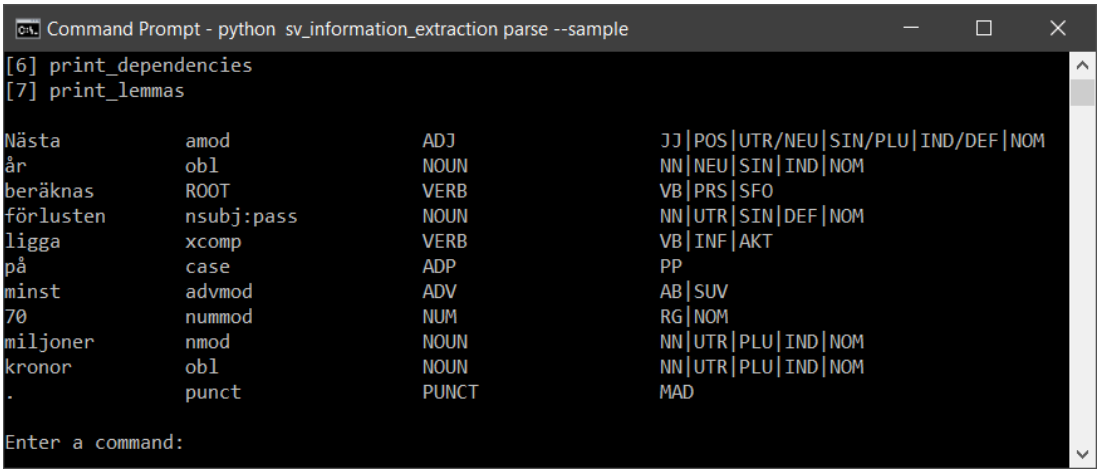

Figure 4.3: Example of the syntactic_info tabular output format

the token, its syntactic dependency, and the dependency head. An example output is shown in Figure 4.4.

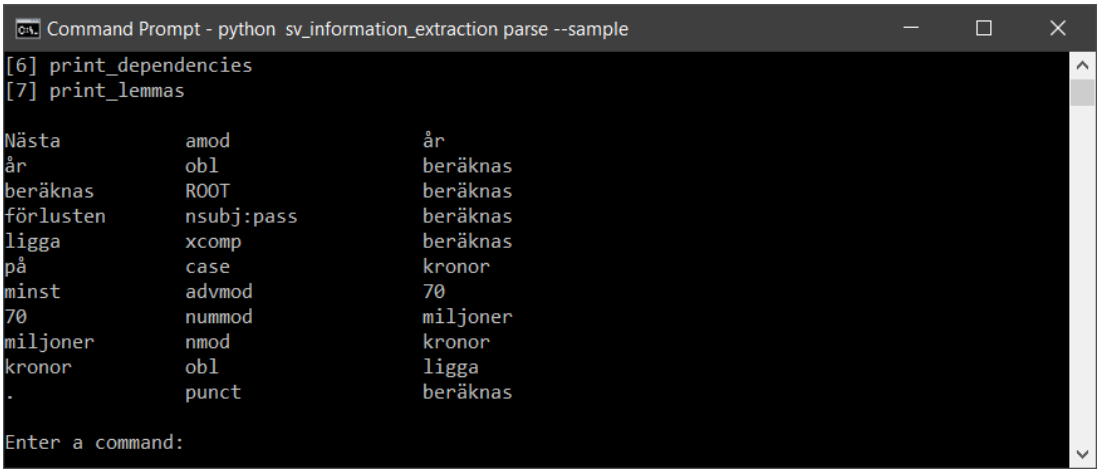

Figure 4.4: Example of the print_dependency_skeleton tabular output format

These dependencies help to visualise the relationships which exist between tokens and are also important for developing patterns that extract phrases. Such patterns are explored in later sections.

4.5 Named Entity Recognition

The NER module is comprised of 12 methods, 2 of which are accessed via the command-line interface. The two aforementioned methods are shown in Figure 4.2 and aim to present NEs in two different formats: an NLTK Tree format, and a simple list format. An example NLTK formatted output is shown in Figure 4.5.

Method	Parameters	Description
split_sents	Plaintext	Splits an input doc into sentences
contains_uppercase	Plaintext	Returns if an input text contains an uppercase character
tagged_sents	spaCy doc	Returns a list of tuples of tokens and their respective POS tags
spacy_ents	spaCy doc	Returns the spaCy named entities
tree_string	List of tuples	Returns an NLTK Tree string representation of the tagged_sents
tree	List of entities, Plaintext	Returns an NLTK Tree of the input doc, with NEs highlighted
tree_labels	NLTK Tree	Returns a list of the POS labels
v2_ents	List of labels, List of tuples	Returns an NLTK Tree of hybrid spaCy and custom entities.
combine_trees	NLTK Tree, NLTK Tree	Returns a combined NLTK Tree that features NEs from both inputs
nltk_ne_trees	Plaintext, spaCy model	Returns an NLTK Tree of all the NEs from combined trees
format_nltk_ents	Plaintext, List of entities	Returns a list of lists. Each list contains the start/end indices, NER label, and NE
spacy_ne	Plaintext, spaCy model	Returns a list of tuples. Each tuple contains the NE and label

Table 4.2: Overview of ie_ner.py functions

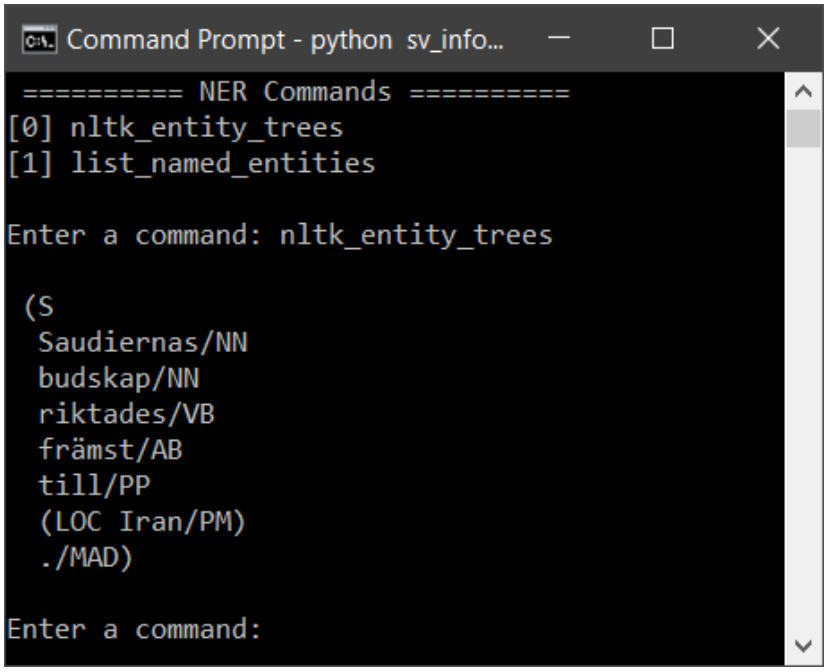

Figure 4.5: Example NLTK Tree representation of a Swedish sentence

The various NER methods and their descriptions are highlighted in Table 4.2, while a selection of some of these methods is covered below.

4.5.1 tagged_sents

tagged_sents returns a list of tuples, each comprising a token and its POS tag. Its method is shown in Figure 4.6. To catch any potential mis-tagging performed by the statistical spaCy POS tagger, regular nouns are updated to a custom “UN” POS tag if the token contains an uppercase character and isn’t the first word in the sentence. For instance, the spaCy POS tagger assigns “ZeniMax Media” the POS tags “PM” and “NN”, rather than "PM" for both. Subsequently, the base spaCy NER system would misrecognise “ZeniMax” as a singular organisation, rather than the actual company name.

```
# Returns a list of tuples of words and their tags
# E.g., return format: [("word", "tag"), ...]
def tagged_sents(doc):
    # loop through words
    for index, token in enumerate(doc):
        upper = contains_uppercase(token.text)
        # if word is a noun, has an uppercase character and is not
        # the first word in the sentence
        if token.tag_.split("|")[0] == "NN" and upper and index != 0:
            token.tag_ = "UN" # assign updated noun tag
    return [(token.text, token.tag_.split("|")[0]) for token in doc]
```

Figure 4.6: Code excerpt from the tagged_sents method

The assigned “UN” POS tag is searched for in the custom pattern-based ruleset. For example, an organisation-matching pattern is displayed in Figure 4.7. In this pattern, 1 or more preceding proper nouns and 0 or more “UN” nouns would be assigned the “ORG” label.

```
pattern = r"""ORG: {<PM>+<UN>*}"""
```

Figure 4.7: ORG assigning pattern example

This pattern works in conjunction with various other patterns to accurately update POS tags and identify any NEs within an input doc.

4.5.2 nltk_ne_trees

nltk_ne_trees returns a list of NLTK Trees, whereby one Tree is constructed for every sentence in an input doc. A code-snippet is illustrated in Figure

4.8. Tokens within input sentences are adjusted to ensure their POS tags are accurate (see Section 4.5.1) before being used to generate an NLTK Tree of entities identified by spaCy. This spaCy Tree is then combined with a custom rule-based Tree to form the NLTK NER Tree which is added to the output list. The constructed Tree objects are then displayed to the user through the relevant command-line command.

```
# Returns a list of NLTK Tree objects for each sentence
# E.g., return format: [Tree1, Tree2, ...]
def nltk_ne_trees(text, nlp):
    sents = split_sents(text)
    trees = []
    for sent in sents:
        doc = nlp(sent)
        tsents = tagged_sents(doc)
        # construct an NLTK Tree with spacy entities
        ne = spacy_ents(doc)
        spacy_tree = tree(ne, tree_string(tsents))
        # extract assigned labels and use in custom
        # NLTK Tree (ran through various patterns)
        labels = tree_labels(spacy_tree)
        v2_tree = v2_ents(labels, tsents)
        # combine Tree objects to form new Tree
        tree_string = combine_trees(spacy_tree, v2_tree)
        new_tree = nltk.tree.Tree.fromstring(tree_string)
        trees.append(new_tree)
    return trees # return Tree objects
```

Figure 4.8: Code excerpt from the nltk_ne_trees method

4.6 Information Extraction

The IE module splits its functionality across 19 methods (see Table 4.3). These methods are used in order to provide 3 command-line functions which are shown in Figure 4.9. The outputs from these commands are displayed using pandas DataFrame objects for readability, with large datasets being accessed through an additional *--sample* argument for testing purposes. Additionally, a percentage is displayed to highlight the number of sentences which matched one of the three command-line phrase patterns.

The module uses Swedish-specific (see Chapter 3) XPOS tags and custom syntactic dependency-based rules to pattern-match phrases. A selection of some of the patterns and algorithms used is covered below.

Method	Parameters	Description
split_sents	Plaintext	Splits an input doc into sentences
tagged_sent_as_str	Plaintext	Converts a POS tagged sentence into non-tagged
is_comp_verb	Plaintext, Index	Returns if a verb has a clausal component
dep_is_in_tree	Dependency, NLTK Tree	Returns if a dependency is in an NLTK Tree
match_modified_noun	Plaintext, Index	Returns a modified noun phrase, e.g., "small dog"
get_subject_phrase	Plaintext, Index	Returns the subject part of a phrase
get_verb_phrase	Plaintext, Index	Returns the verb part of a phrase
get_object_phrase	Plaintext, Verb Phrase, Index	Returns the object part of a phrase
match_verb_phrases	Plaintext	Returns verb phrases which follow a SVO format, including any auxiliaries/clauses
match_descriptive_nouns	Plaintext	Returns noun phrases where an adjective modifies the noun
modify_noun	Plaintext, Index	Returns noun phrases which are compound
match_preposition_nouns	Plaintext	Returns noun phrases where there is a preceding preposition
func_as_str	Key	Returns a key string
results_as_pandas_df	List, Key, spaCy model, Boolean	Returns a pandas df object for the output data for the three main functions
output_percentage	pandas DF, Columns	Returns the percentage of rows matched
setup_pandas	None	Assigns width/height for pandas dataframe
pandas_df	Boolean, List, Key, spaCy model	Returns the pandas df object as a function specific formatted output
extract_info	List, Key, spaCy model	Prints the output for the function entered as the key string
extract_info_sampleset	List, Key, spaCy model	Prints the output for the function entered as the key string

Table 4.3: Overview of ie_sv.py functions

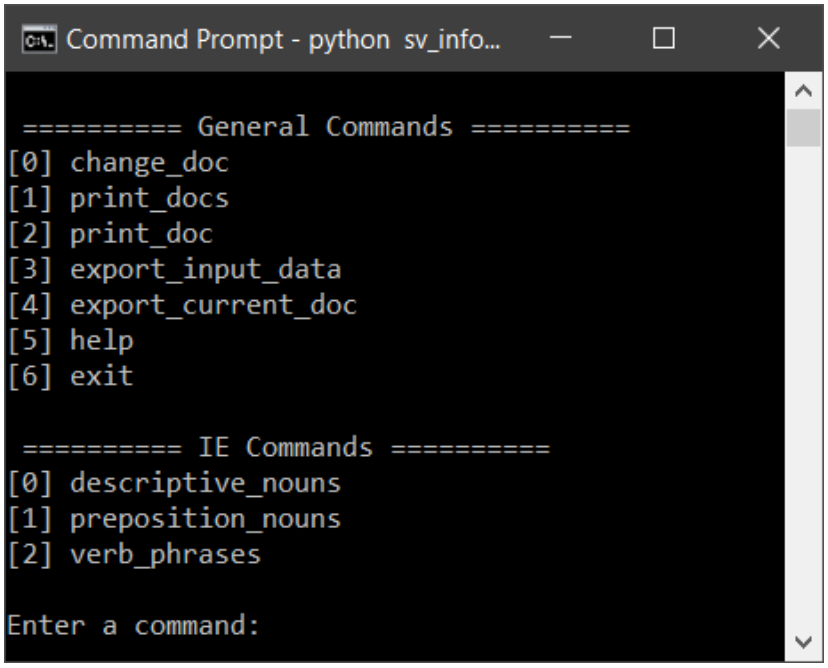

Figure 4.9: Information Extraction module command-line interface

4.6.1 get_subject_phrase

get_subject_phrase returns the subject phrase for a verb phrase. A pseudo-code run-through is shown in Algorithm 1. The algorithm takes an input root

verb and attempts to locate the sentence subject, including any modifiers.

$token \leftarrow text[index]$

$o\_phrase \leftarrow None$
$nouns \leftarrow [\prime NOUN\prime, \prime PROPN\prime, \prime PRON\prime]$
for $right \in token.rights$ **do**
  if $right.pos\_ = \prime VERB\prime$ *and* $right.dep\_ \in [\prime ccomp\prime, \prime xcomp\prime]$ **then**
    $m\_phrase \leftarrow None$
    for $child \in right.children$ **do**
      if $child.dep\_ = \prime mark\prime$ **then**
        $o\_phrase \leftarrow child.text$
      else if $child.dep\_ = \prime obj\prime$ *and* $len(m\_phrase) = 0$ **then**
        $nbor \leftarrow child.nbor(-1)$
        if $nbor.pos\_ = \prime ADP\prime$ *and* $nbor.dep\_ = \prime case\prime$ **then**
          $v\_phrase \leftarrow nbor.text$
        $o\_phrase \leftarrow child.text$
      else if $child.pos\_ = \prime ADV\prime$ *and* $child.dep\_ = \prime advmod\prime$ **then**
        $v\_phrase \leftarrow child.text$
    end
    $v\_phrase \leftarrow m\_phrase + right.text$
  else if $right.pos\_$ *in* $nouns$ *and* $right.dep\_$ *in* $[\prime obj\prime, \prime obl\prime]$ **then**
    `/* calculating additional components of the verb phrase */`
end

Algorithm 1: Psuedo-code for the get_object_phrase function

4.6.2 match_descriptive_nouns

match_descriptive_nouns returns a list of descriptive noun phrases for a sentence. The algorithm is illustrated as pseudo-code in Algorithm 2 and works by first locating a noun which is either a subject, object, etc. The noun's syntactically dependent children are then traversed and any modifiers, such as adjectives, are appended as a noun phrase. The noun phrase

is appended to the list and the list returned to be displayed to the user.

```
matches ← None

for token ∈ text do
    phrase ← None

    deps ← ['obj','nsubj','nsubj:pass','csubj','obl']
    if token.pos_ = 'NOUN' and token.dep_ ∈ deps then
        for sub_token ∈ token.children do
            if sub_token.pos_ = 'ADJ' and sub_token.dep_ = 'amod' then
                phrase ← sub_token.text
            end

            if len(phrase) != 0 then
                phrase ← token.text
            end
        end
    end

    if len(phrase) != 0 then
        matches.append(phrase)
    end
end
return matches
```

Algorithm 2: Psuedo-code for the match_descriptive_nouns function

4.7 Summary and Discussion

In this chapter, the implementation of a novel Swedish information extraction tool was outlined. The important aspects include:

- The various libraries and technologies which the tool is dependent on to function, such as spaCy, NLTK, etc.

- The nature of the tool, i.e., it is a command-line implementation rather than an importable Package.

- The various modules and systems which comprise the tool and details about how they work.

Other useful aspects of the implementation were also covered, such as the project file structure, resources used, implementation difficulties, etc. The information presented in this chapter is sufficient to ensure that the tool can be installed and run quickly by the user.

Chapter 5

Evaluation

5.1 Introduction

This chapter compares the results of the four main information extraction implementations: custom NER, descriptive_nouns, preposition_nouns, and verb_phrases. The standard evaluation metrics of Precision, Recall, and F-Measure are used to compare the systems. The following section features an overview of the systems, as well as the test-set being used for evaluation. Each subsequent section provides an overview of the performance for each individual system, with a comprehensive discussion of each.

5.2 Evaluation Overview

Evaluating the performance of the four main systems required their outputs to be manually checked against a desired test-set. For the custom NER, this was relatively simple as the SUC 3.0 corpus features NEs which are already annotated. However, the three IE systems required manual test-sets to be constructed. This process is relatively time-consuming, so it was decided that 100 randomly selected sentences should be used for the evaluation of each system. An overview on the test-sets is shown in Table 5.1.

System	Sentences	Tokens	Potential
descriptive_nouns	100	1956	157
preposition_nouns	100	1532	150
verb_phrases	100	1543	116
Custom NER	100	1724	234

Table 5.1: Overview of the test-set

Command	Dataset Size	Overall Match
descriptive_nouns	20,000	38.88%
preposition_nouns	20,000	36.30%
verb_phrases	20,000	52.47%

Table 5.2: Overview of System Match Percentage

Table 5.2 shows a comparison of the percentage of matched sentences for each system. Comparing this information with Table 5.1 yields a useful overview into the nature of the selected test-set. The verb_phrases system matches a higher proportion of sentences, yet features a lower number of potentially match-able phrases. As such, it can be inferred that there are more sentences with at least one verb phrase, but each sentence features comparatively less phrases than the other systems. This highlights the distribution nature of sources within SUC 3.0, with sentence contents depending on the context of the text (i.e., fiction, non-fiction, etc.).

5.2.1 Overall Results Comparison

A comparison of the performance of the main IE systems is shown in Figure 5.1. The descriptive_nouns system performed the best of the three systems, with verb_phrases performing worst. However, verb_phrases also yielded the closest Precision and Recall scores, making it the most consistent of the three systems.

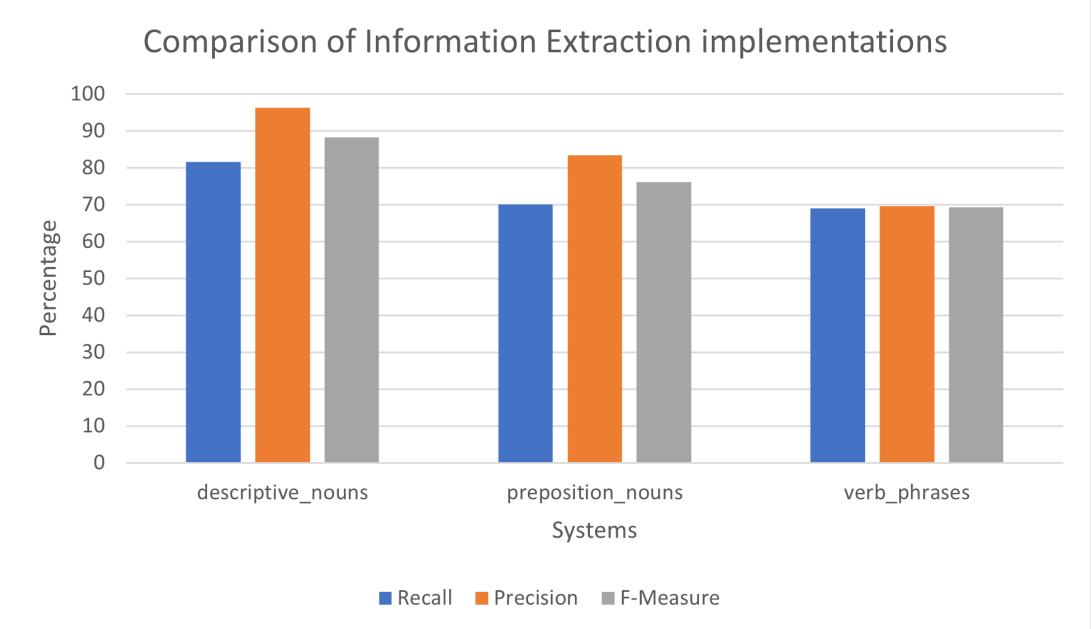

Figure 5.1: IE implementation system comparison

System	Recall	Precision	F-Measure
descriptive_nouns	81.53	96.24	88.28
preposition_nouns	70.00	83.33	76.09
verb_phrases	68.97	69.57	69.26
Custom NER	94.17	87.39	90.65

Table 5.3: System Evaluation Metric Comparison

Table 5.3 displays a tabular comparison of the Precision, Recall, and F-Measure of the four main systems, including the custom NER. The custom NER system yielded the highest recall of any system, but featured a comparatively low precision. This is opposite to the next best performing system, descriptive_nouns, which featured a comparatively higher precision.

5.3 Custom NER Performance Overview

As described in Chapter 3, the custom NER system is a hybrid approach which utilises a pre-trained statistical model in conjunction with a rule-based approach. The pre-trained model was trained with SUC 3.0 and yielded a recall of 84.48, a precision of 86.27, and an F-Measure of 85.37. Comparatively, the custom NER system achieved a recall of 94.17, a precision of 87.39, and an F-Measure of 90.65.

5.3.1 Discussion

A graphical breakdown of the assigned NER categories is shown in Figure 5.2.

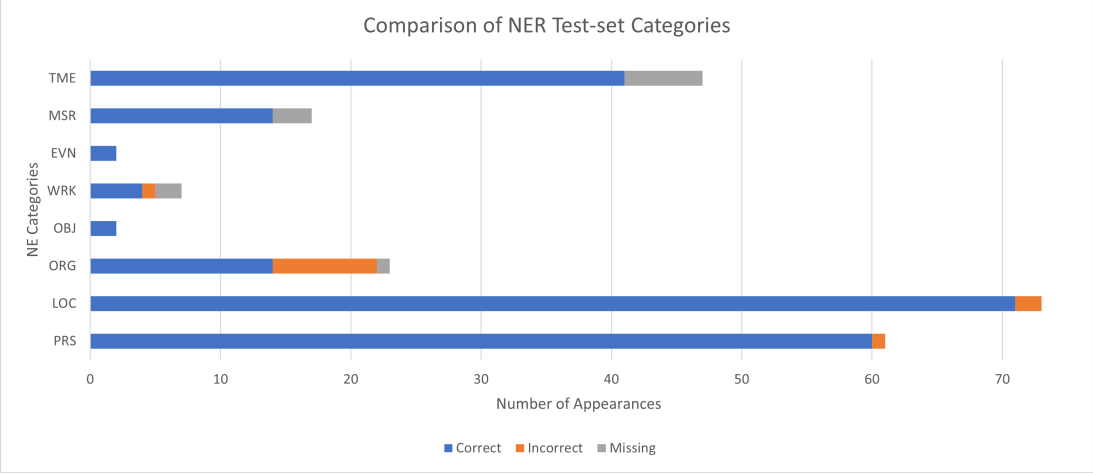

Figure 5.2: NER implementation category distribution

Combination	Total
Correct Category & Correct NE	194
Correct Category & Incorrect NE	13
Incorrect Category & Correct NE	10
Incorrect Category & Incorrect NE	5
Missing NEs	12

Table 5.4: NE Category and Content Analysis

Entities which fit under the LOC, PRS, and TME categories were most prevalent in the test-set, with EVN and OBJ being the least common. Furthermore, of the NE categories, ORG features the most inaccuracy. These inaccuracies are explored further in Table 5.4, which shows that there is an even distribution of inaccuracy between categorisation and NE extraction.

Overall, the custom NER system outperforms the standalone statistical model, particularly with relation to NE identification. Furthermore, as a standalone solution, the custom NER system would perform well against the various existing Swedish NER approaches (see Chapter 3).

5.4 descriptive_nouns Performance Overview

The descriptive_nouns system achieved a recall of 81.53, a precision of 96.24, and an F-Measure of 88.28. Of the three main IE systems, descriptive_nouns featured the best performance and was the most accurate.

5.4.1 Discussion

There is a noticeable contrast between the recall and precision values, which highlights the system’s tendency to miss valid noun phrases rather than incorrectly identify invalid ones. This is corroborated through analysis of the system’s output which shows that implied noun descriptions and compound words are not recognised.

One such implied noun description exists for the noun "landskapet", i.e. "landskapets oändliga skönhet" (the infinite beauty of the landscape). This format is not recognised by the system as it only accounts for the target

structure ADJECTIVE-NOUN, rather than an implied description which follows the structure NOUN-ADJECTIVE.

However, despite this missing functionality, the system performs adequately and extracted most of the test-set descriptive noun phrases. Furthermore, the changes required to ensure the NOUN-ADJECTIVE format are relatively simple due to the rule-based nature of the implementation.

5.5 preposition_nouns Performance Overview

The preposition_nouns system achieved a recall of 70.0, a precision of 83.3, and an F-Measure of 76.09. The main aim of the system was to extract nouns which were preceded by a preposition, such as "i huset" (in the house).

5.5.1 Discussion

This system performed as expected for most sentences, but noticeably missed most modifiers for nouns, including numbers. Furthermore, the system was restricted to a SUBJECT-PREPOSITION-OBJECT format, which excluded the PREPOSITION-OBJECT and VERB-PREPOSITION-NOUN formats.

One incorrect output by the system was: "mil utanför Vetlanda" (miles outside Vetlanda). This is grammatically correct and fits the structure, but is missing the modifier "några" (some). In some scenarios this is insignificant, but in this context the phrase changes without it. The decision on whether this output is incorrect was difficult to make, as it could be argued that the descriptive_nouns system is responsible for such an extraction. The three IE systems were designed to work in tandem with each other, so a blurred distinction exists between the expected correct output.

Overall, the system performance is adequate and the problems can be rectified relatively easily through new rule-based patterns. However, the presented issues highlighted a potential need for combining the IE implementation systems (see Chapter 6).

5.6 verb_phrases Performance Overview

The verb_phrases system achieved a recall of 68.97, a precision of 69.57, and an F-Measure of 69.26. Of the three main IE systems, verb_phrases performed the worst, but was also the most consistent.

5.6.1 Discussion

As explored in preposition_nouns, this system missed certain modifiers, descriptors, etc. which were important for the grammatical/completeness of the system. However, as discussed in Section 5.5.1 and Chapter 6, all three systems were designed to be used in conjunction with each other, so there is a slightly blurred distinction for the correct output.

One incorrect output by the system was: "som utför arbeten" (who performs the work), which lacks "kyrkliga textila". These missed modifiers change the sentence to: "som utför kyrkliga textila arbeten" (performing church textile works). The descriptive_nouns system could effectively extract these modifiers, but the verb_phrases system follows a strictly NOUN-VERB-NOUN format.

Overall, this system was the most difficult to implement and this fact is reflected in its performance. However, combining the various IE implementation systems (see Chapter 6) is relatively simple and would improve the performance.

5.7 Summary and Discussion

In summary, all four of the implemented systems have performed adequately. On average, the systems achieved an F-Measure of 81.07, with no individual system having an F-Measure less than 65. This is noticeably higher than researched systems, such as FASTUS (see Section 2.7.2), but still leaves room for improvement.

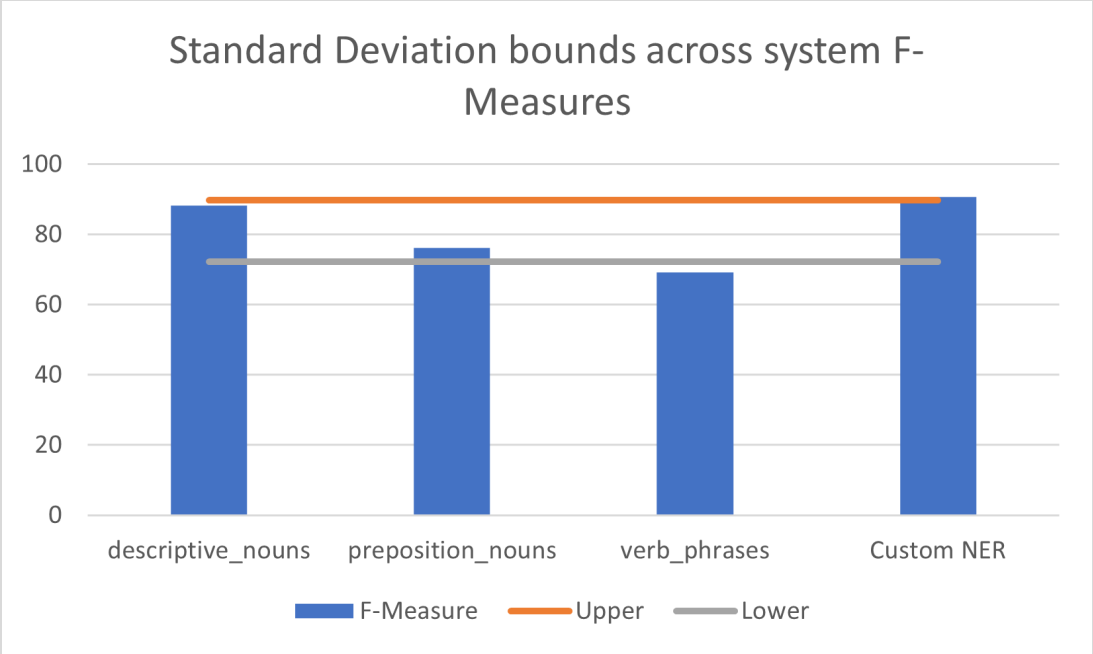

Figure 5.3: Standard Deviation bounds for implemented system F-Measures

As shown in Figure 5.3, both the custom NER and descriptive_nouns systems operate at almost one Standard Deviation above the average of the four systems. As such, it can be inferred that these systems perform exceptionally well. However, the verb_phrases system operates at a level which is greater than one Standard Deviation lower than the average. This statistic highlights the variance in the performance between the four systems, further outlining the future improvements which could be made. These potential improvements are explored further in Chapter 6.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

The original aim of this dissertation was to create a novel Python tool which performs information extraction on Swedish text. A manual evaluation of the implementation's results showed that relevant and accurate information is extracted as expected. Furthermore, the system extracts a variety of different information structures, such as verb phrases, described nouns, and preposition-bound nouns. In addition to the overall aim, this dissertation also featured three research-specific objects: to review relevant NLP literature about IE, to implement IE as a Python tool, and to perform an evaluation on the implementation. Each of these objectives have been met in sections 2, 3 and 4, and 5, respectively.

While the implementation does perform information extraction on Swedish text, it does so by splitting functionality across various command-line methods. This approach is effective for some use-cases, but does prove difficult or disadvantageous in others. Due to functionality being split across command-line methods, some extracted structures lack depth of information. This can occasionally provide grammatically incorrect or confusing outputs, but is rectified by using the systems in tandem for an input doc. An example scenario is given in Chapter 5.

Overall, the system performs as expected and fulfils the original objectives. The ideal end-goal for this implementation would be to achieve a near perfect performance, thus achieving the highest quality solutions.

6.2 Review of Aim and Objectives

6.2.1 Dissertation Aim

The overall dissertation aim was to create a novel tool which implements a method for information extraction of Swedish text using Python.

By presenting a tool which features various methods for information extraction of Swedish text, it can be confidently stated that the overall aim of the dissertation has been met. The presented tool identifies named entities and extracts descriptive noun phrases, preposition-bound noun phrases, and verb phrases. These methods are distributed as a command-line Python tool, with potential to also be distributed as a Package.

6.2.2 Research-specific Objectives

To ensure that the overall aim was attained, three research-specific objectives were outlined at the beginning of the dissertation. These objectives are:

1. To review the relevant literature in the Natural Language Processing field, with specific reference to Information Extraction.

2. To implement Information Extraction with Swedish text as a command-line Python tool.

3. To compare the implementation and results against the training set.

The first objective is completed within Chapter 2, where various aspects of the information extraction task are covered. Some of the covered topics include extraction techniques, output formats, and statistical modelling. Existing systems, such as FASTUS, are also explored in depth.

The second objective is completed across Chapters 3 and 4. A comprehensive analysis of Swedish NLP is completed, whereby any existing NLP resources are compared and discussed. Furthermore, as outlined in 6.2.1, a tool

which features various methods for information extraction of Swedish text is implemented.

The third objective is completed within Chapter 5, whereby a manual evaluation of the implementation's results showed that the information extracted by the system is relevant and accurate. In addition, some of the implemented systems demonstrate noticeable improvements over existing solutions. These results are encouraging for the presented implementation and corroborate that the research-specific objectives have been met.

6.3 Future Work

Some of the possible improvements which could be made include:

- Improve the code efficiency and structure
- Combine the implemented IE systems into a single Python Package
- Extend the functionality – allow doc and module changing, etc.
- Allow for models to be trained using the tool

6.3.1 Code Efficiency and Structure

As discussed in Chapter 3, Python is a programming language which features various best practices which new developers may not be familiar with. A larger effort could be made to adhere to these practices, i.e. a more considerate choice of data structures, use of the available Object Oriented paradigm, etc.

In addition, a recursive approach could be utilised for the current iterative methods used in the IE systems. This approach would be particularly compatible with the IE use-case, as it involves navigating left and right sub-trees that can vary in length.

6.3.2 Combining Systems as a Package

The various IE functionalities are currently split across three command-line functions – descriptive_nouns, preposition_nouns, and verb_phrases. The tool is designed to be used such that the three functionalities are used in conjunction with one another, but this approach can lead to confusing outputs and essentially reduces efficiency. Merging or combining the systems into a single Python Package would make the process more streamlined and the outputs more meaningful.

6.3.3 Functionality Extensions

Currently, the user experience is marred by missing functionality which often obstructs the user from completing tasks seamlessly. For example, the user cannot change the input text source or navigate between the different modes without first restarting the tool. Implementing changes to provide this functionality would allow the user to spend less time navigating the UI and also make the experience feel more streamlined, thus increasing the overall ergonomics of the tool.

6.3.4 Model Training within the Tool

The tool currently allows for models to be inserted directly into the source files (see Chapter 3), but doesn't support the training of new models. This can be completed relatively easily through the existing spaCy pipeline, provided a suitably sized corpora is provided. Furthermore, the existing file structure is already designed to allow for corpora and other training resources to be loaded, so the inclusion of the added functionality wouldn't require any significant changes. By implementing this functionality, the tool would be a more complete package and an invaluable resource for Swedish NLP.

References

[1] D. Appelt, J. Hobbs, J. Bear, D. Israel and M. Tyson, ‘Fastus: A finite-state processor for information extraction from real-world text.,’ Jan. 1993, pp. 1172–1178 (pp. 8, 13–15).

[2] D. E. Appelt, ‘Introduction to information extraction,’ *Ai Communications*, vol. 12, no. 3, pp. 161–172, 1999 (pp. 3, 4, 6).

[3] M. Baroni, ‘39 distributions in text,’ *Corpus Linguistics: An International Handbook Volume*, vol. 2, pp. 803–822, 2005 (pp. 10, 11).

[4] C. Biemann, C. Giuliano and A. Gliozzo, ‘Unsupervised part-of-speech tagging supporting supervised methods,’ in *Proceedings of Recent Advances in Natural Language Processing*, 2007 (p. 9).

[5] S. Bird *et al.*, ‘Multidisciplinary instruction with the natural language toolkit,’ Association for Computational Linguistics, 2008 (pp. 19, 20).

[6] S. Bird, E. Klein and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O’Reilly Media, Inc.", 2009 (pp. 1, 4, 6, 7, 9, 20, 34).

[7] L. Borin, M. Forsberg, M. Hammarstedt, D. Rosén, R. Schäfer and A. Schumacher, ‘Sparv: Språkbanken’s corpus annotation pipeline infrastructure,’ in *The Sixth Swedish Language Technology Conference (SLTC), Umeå University*, 2016, pp. 17–18 (p. 22).

[8] L. Borin, M. Forsberg and J. Roxendal, ‘Korp-the corpus infrastructure of spräkbanken.,’ in *LREC*, 2012, pp. 474–478 (p. 22).

[9] E. Brill, ‘A simple rule-based part of speech tagger,’ PENNSYLVANIA UNIV PHILADELPHIA DEPT OF COMPUTER and INFORMATION SCIENCE, Tech. Rep., 1992 (p. 24).

[10] C. Cardie, ‘Empirical methods in information extraction,’ *AI magazine*, vol. 18, no. 4, pp. 65–65, 1997 (p. 12).

[11] N. Chinchor and B. M. Sundheim, ‘Muc-5 evaluation metrics,’ in *Fifth Message Understanding Conference (MUC-5): Proceedings of a Conference Held in Baltimore, Maryland, August 25-27, 1993*, 1993 (pp. 13, 14).

[12] J. D. Choi, J. Tetreault and A. Stent, ‘It depends: Dependency parser comparison using a web-based evaluation tool,’ in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2015, pp. 387–396 (p. 21).

[13] K. Clark and C. D. Manning, ‘Improving coreference resolution by learning entity-level distributed representations,’ in *Association for Computational Linguistics (ACL)*, 2016. [Online]. Available: `https://nlp.stanford.edu/pubs/clark2016improving.pdf` (p. 7).

[14] M.-C. De Marneffe and C. D. Manning, ‘Stanford typed dependencies manual,’ Technical report, Stanford University, Tech. Rep., 2008 (p. 26).

[15] C. Djoweini and H. Hellberg, *Approaches to natural language processing in app development*, 2018 (pp. 19–21).

[16] G. Doddington, A. Mitchell, M. Przybocki, L. Ramshaw, S. Strassel and R. Weischedel, ‘The automatic content extraction (ACE) program – tasks, data, and evaluation,’ in *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC’04)*, Lisbon, Portugal: European Language Resources Association (ELRA), May 2004. [Online]. Available: `http://www.lrec-conf.org/proceedings/lrec2004/pdf/5.pdf` (p. 5).

[17] D. M. Eberhard, G. F. Simons and C. D. Fennig (eds.), *Ethnologue: Languages of the world. twenty-fourth edition.* [Online]. Available: `http://www.ethnologue.com/` (p. 17).

[18] T. Eftimov, B. Koroušić Seljak and P. Korošec, ‘A rule-based named-entity recognition method for knowledge extraction of evidence-based dietary recommendations,’ *PloS one*, vol. 12, no. 6, e0179488, 2017 (p. 25).

[19] R. Grishman, ‘Information extraction: Techniques and challenges,’ in *International summer school on information extraction*, Springer, 1997, pp. 10–27 (pp. 3, 5–8).

[20] R. Grishman and B. Sundheim, 'Message Understanding Conference-6: A brief history,' in *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, 1996. [Online]. Available: https://www.aclweb.org/anthology/C96-1079 (pp. 4, 13–15).

[21] M. Gritta, M. T. Pilehvar and N. Collier, 'A pragmatic guide to geoparsing evaluation: Toponyms, named entity recognition and pragmatics,' eng, *Language resources and evaluation*, vol. 54, no. 3, pp. 683–712, 2020, ISSN: 1574-020X (p. 4).

[22] S. Gustafson-Capková and B. Hartmann, 'Manual of the stockholm umeå corpus version 2.0,' *Unpublished Work*, 2006 (pp. 28, 29).

[23] J. R. Hobbs *et al.*, 'Fastus: A cascaded finite-state transducer for extracting information from natural-language text,' *Finite-state language processing*, pp. 383–406, 1997 (pp. 14–16, 35).

[24] N. Indurkhya and F. J. Damerau, *Handbook of natural language processing*. CRC Press, 2010, vol. 2 (pp. 6, 8–11).

[25] J. Johnson, *Countries with the highest internet penetration rate 2021 | statista*, 2021. [Online]. Available: https://www.statista.com/statistics/227082/countries-with-the-highest-internet-penetration-rate/ (p. 1).

[26] D. Kokkinakis, J. Niemi, S. Hardwick, K. Lindén, L. Borin *et al.*, 'Hfst-swener–a new ner resource for swedish,' in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, European Language Resources Association (ELRA), 2014 (p. 29).

[27] W. Lehnert *et al.*, 'Umass/hughes: Description of the circus system used for muc-5.,' Jan. 1993, pp. 277–291. DOI: 10.1145/1072017.1072043 (p. 8).

[28] M.-C. de Marneffe *et al.*, 'Universal dependencies: A cross-linguistic typology,' (p. 3).

[29] M. Mohri, A. Rostamizadeh and A. Talwalkar, *Foundations of machine learning*. MIT press, 2018 (p. 9).

[30] D. Nadeau and S. Sekine, 'A survey of named entity recognition and classification,' *Lingvisticae Investigationes*, vol. 30, Jan. 2007. DOI: 10.1075/li.30.1.03nad (p. 15).

[31] J. Nilsson, J. Hall and J. Nivre, 'Mamba meets tiger: Reconstructing a swedish treebank from antiquity,' *Proceedings from the special session on treebanks at NODALIDA 2005*, pp. 119–132, 2005 (p. 30).

[32] J. Nivre *et al.*, 'Universal dependencies v1: A multilingual treebank collection,' in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, 2016, pp. 1659–1666 (pp. 26, 27).

[33] R. Östling, 'Stagger: An open-source part of speech tagger for swedish,' *Northern European Journal of Language Technology*, vol. 3, pp. 1–18, 2013 (pp. 1, 17, 19, 22, 24, 28).

[34] S. Patel, V. Dabhi and H. Prajapati, 'Extractive based automatic text summarization,' *Journal of Computers*, vol. 12, p. 550, Nov. 2017. DOI: 10.17706/jcp.12.6.550-563 (pp. 5, 6).

[35] S. Petrov, D. Das and R. McDonald, 'A universal part-of-speech tagset,' in *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey: European Language Resources Association (ELRA), May 2012, pp. 2089–2096. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2012/pdf/274_Paper.pdf (p. 26).

[36] P. Qi, Y. Zhang, Y. Zhang, J. Bolton and C. D. Manning, 'Stanza: A python natural language processing toolkit for many human languages,' *arXiv preprint arXiv:2003.07082*, 2020 (p. 23).

[37] E. Roche and Y. Schabes, *FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text*. 1997, pp. 383–406 (pp. 1, 3, 4).

[38] R. Rosenfeld, 'A maximum entropy approach to adaptive statistical language modeling,' 1996 (p. 10).

[39] S. Sarawagi, *Information extraction*. Now Publishers Inc, 2008 (pp. 4–8, 12, 26).

[40] S. språknämnden, *Språkriktighetsboken*, ser. Skrifter utgivna av Svenska språknämnden. Norstedts Akademiska Förlag, 2005, ISBN: 9789172273818. [Online]. Available: https://books.google.co.uk/books?id=kvtFSAAACAAJ (p. 17).

[41] R. K. Srihari, W. Li, T. Cornell and C. Niu, ‘Infoxtract: A customizable intermediate level information extraction engine,’ *Natural Language Engineering*, vol. 14, no. 1, pp. 33–69, 2008 (pp. 1, 3).

[42] W. J. Teahan, ‘A compression-based toolkit for modelling and processing natural language text,’ *Information*, vol. 9, no. 12, p. 294, 2018 (p. 19).

[43] E. F. Tjong Kim Sang, ‘Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition,’ in *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*, 2002. [Online]. Available: https://www.aclweb.org/anthology/W02-2024 (p. 15).

[44] E. F. Tjong Kim Sang and F. De Meulder, ‘Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition,’ in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 142–147. [Online]. Available: https://www.aclweb.org/anthology/W03-0419 (p. 15).

[45] J. VanderPlas, *Python data science handbook: Essential tools for working with data*. " O’Reilly Media, Inc.", 2016 (pp. 19, 21).

[46] Y. Vasiliev, *Natural Language Processing with Python and SpaCy: A Practical Introduction*. No Starch Press, 2020 (pp. 9, 21).

[47] Y. Wang *et al.*, ‘Clinical information extraction applications: A literature review,’ *Journal of Biomedical Informatics*, vol. 77, pp. 34–49, 2018, ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2017.11.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046417302563 (pp. 1, 3, 4, 35).

Appendix A

Poster

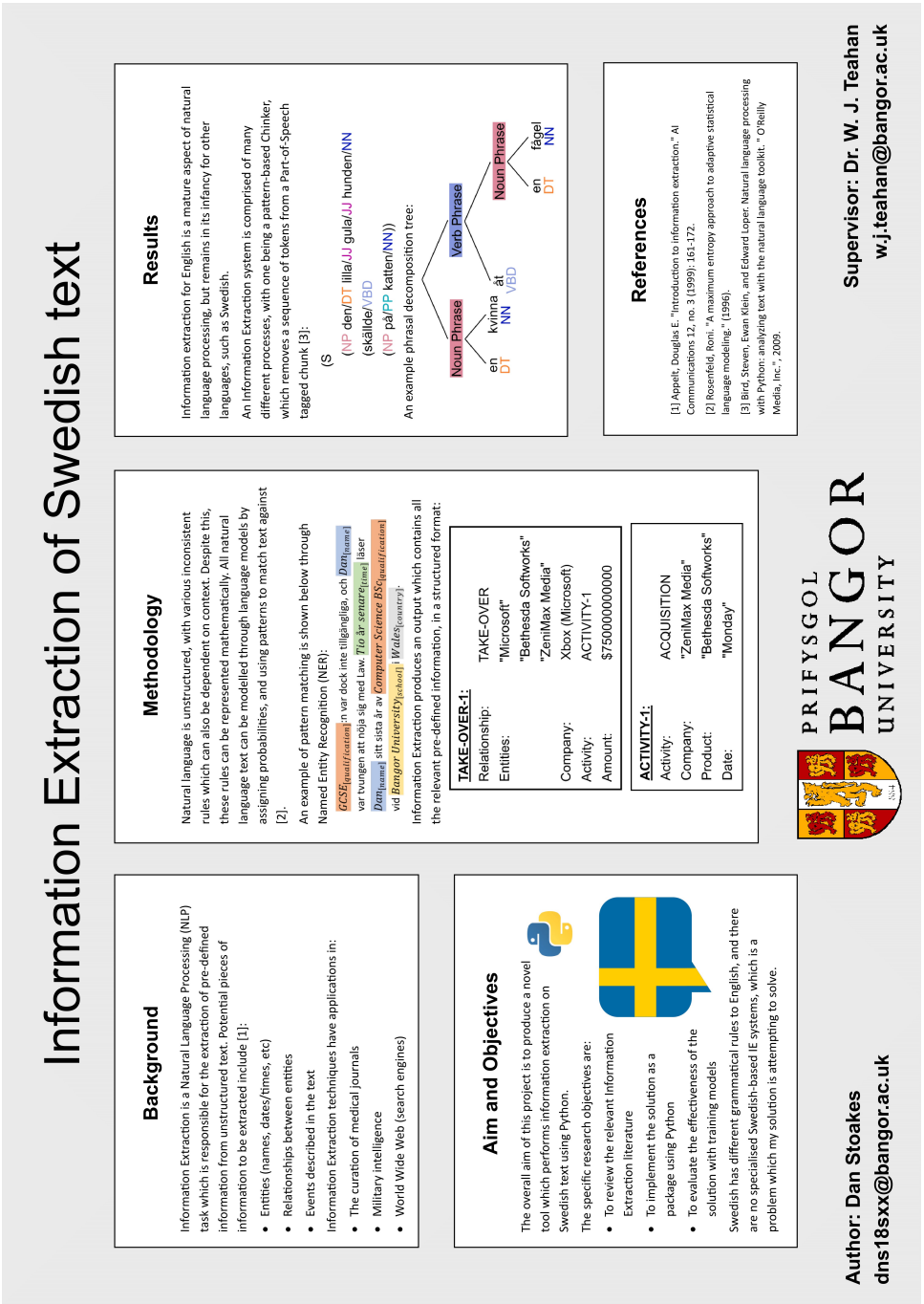

Figure A.1: Poster for the Information Extraction of Swedish text