GOETHE UNIVERSITY FRANKFURT

Institute for Computer Science

Bachelor Thesis

Natural Language Processing to enable semantic search on numismatic descriptions

Patricia Klinger

February 8, 2018

supervised by
Prof. Dott.-Ing. Roberto V. Zicari

Databases and Information Systems (DBIS)

Erklärung zur Abschlussarbeit gemäß § 25, Abs. 11 der Ordnung für den Bachelorstudiengang Informatik vom 06. Dezember 2010:

Hiermit erkläre ich, dass ich die Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe.

Frankfurt am Main, den 8. Februar 2018

Patricia Klinger

Contents

List of Figures

List of Tables

1 Introduction

The numismatic database *Corpus Nummorum Thracorum (CNT)*[1] contains a virtual meta-collection of ancient coins of Thrace. Thrace is an ancient region which covers modern Bulgaria, Northern Greece, and European Turkey over a period from 666 BC to 400 AD. The meta-collection consists of Thracian coins located in museums and private collections from all over the world. By analyzing these coins, coin dies have been reconstructed if possible. The goal of CNT is to generate a typology of Thracian coins[2]. Unique identifiers shall connect the detected types to Nomisma[3]. Nomisma is a project to provide stable digital representations of numismatic concepts according to the principles of Linked Open Data.

The iconography of the ancient coins is described in human language. A description of the reverse and obverse of each coin is given both in English and German. Based on the CNT database dump from January 9, 2018, there exist 3.628 iconographic descriptions, in the following called *designs*. All designs are stored in a relational MySQL database. The designs can be searched, e.g. for a person, its function or its epoch in the CNT Advanced Search[4]. Such queries are based on *key words* which can be selected from a list. For

> "Apollo seated left on omphalos, holding bow in right hand.",

the coherent key words are the name "Apollo" and the object "bow" (cf. section 2.1).

Together with the designs, these key words are stored in the database and contain relevant information. However, they have been extracted manually from the designs. As manual labor is expensive, it would be better to extract information automatically. In *information extraction*, five tasks are distinguished (cf. section 3.1). The most important tasks are *named entity recognition (NER)* and *relation extraction (RE). Named entities (NEs)* are definite noun phrases, e.g. names like "Apollo", that refer to specific types of individuals, e.g. persons, organizations or locations (Bird, Klein, and Loper 2009, chap. 7.5). Extracting the relation between NEs means to built triples of the form (NE_1, α, NE_2). α is the string of words intervening between the two NEs (Bird, Klein, and Loper 2009, chap. 7.6). NER is a challenging learning problem, because usually only a small amount of training data is available (Lample et al. 2016). RE is even more difficult because language ambiguity forms a massive obstacle (Konstantinova 2014, p.24).

Reducing manual labor and enhancing numismatic search would thus be of great help for the numismatic community. Therefore the idea arose to build an application that recognizes, for example, persons in designs. Ideally, the application should be a smart system, whose recognition works automatically and which even recognizes new persons in new designs. Achieving an *accuracy* of below 100% seemed sufficient, because instead of being 100% accurate the application was supposed to generalize from the CNT data to other numismatic data sets (for the definition of accuracy cf. section 3.4). Additionally, it seemed desirable to be able to extract certain relations automatically.

[1]https://www.corpus-nummorum.eu

[2]https://www.corpus-nummorum.eu/about.php

[3]http://www.nomisma.org/

[4]https://www.corpus-nummorum.eu/AdvSearch.php

Definition of task The goal of this bachelor thesis is to take first steps in accessing relevant information in the CNT designs by using *Natural Language Processing (NLP)*, i.e. computer manipulation of natural language (Bird, Klein, and Loper 2009, Preface). The thesis' approach is supposed to reduce manual labor and generalize from the CNT designs to other numismatic data sets. This thesis aims at laying the foundations for a smart search on numismatic descriptions. Consider again the design:

> "Apollo seated left on omphalos, holding bow in right hand."

If all persons holding a bow had to be found, a smart search would find "Apollo". By using NLP, the NEs persons, animals, plants and objects shall be automatically recognized and relations between persons and objects shall be automatically extracted.

For computer programs, human language is not easy to access. Thus it is difficult to extract relevant information from texts. NLP is a specialized field of computer science that tries to deal with this problem. It focuses on building applications to enable interaction between machines and natural languages (Sarkar 2016, p. 46). NLP often employs machine learning techniques and many machine learning libraries, such as scikit learn[5] and pandas[6], use the programming language Python. Therefore all implementations in this thesis are realized with Python.

One possibility to work on human language data with Python is the *Natural Language Toolkit (NLTK)*[7]. Despite NLTK being widely used in academia, it makes it difficult to add new entity types; an important feature given that the CNT designs talk about ancient deities and mythical animals which require new labels. The spaCy NLP library[8] offers the possibility to annotate NEs in natural language texts (cf. section 3.6). Adding new entity types with spaCy is straightforward. Moreover, the interaction with scikit learn and pandas is well established. Therefore this thesis implements NLP with spaCy.

Structure The following chapters provide a more detailed discussion of the briefly sketched approach of this thesis. The second chapter locates the CNT database in numismatics. The third chapter focuses on how information extraction works in general and what methods have been used for meeting the requirements. In chapter four, the implementation of the approach is presented and explained. In chapter five, problems that arouse during implementation are pointed out and the corresponding solutions are discussed. Chapter six leaves room for evaluating the implementation and discussing the results. Finally, chapter seven summarizes the main findings and offers an outlook on further research.

[5]http://scikit-learn.org/
[6]https://pandas.pydata.org/
[7]http://www.nltk.org/
[8]https://spacy.io/

2 Corpus Nummorum Thracorum in Numismatics

2.1 The CNT Database

The numismatic database *Corpus Nummorum Thracorum (CNT)* contains a virtual meta-collection of ancient coins of Thrace. Thrace is an ancient region which covers modern Bulgaria, Northern Greece, and European Turkey over a period from 666 BC to 400 AD. The meta-collection consists of Thracian coins located in museums and private collections from all over the world. By analyzing these coins, coin dies have been reconstructed if possible. The goal of CNT is to generate a typology of Thracian coins. Unique identifiers shall connect the detected types to Nomisma. Nomisma is a project to provide stable digital representations of numismatic concepts according to the principles of Linked Open Data.

The iconography of the ancient coins is described in human language. A description of the reverse and obverse of each coin is given both in English and German. Based on the CNT database dump from January 9, 2018, there exist 3.628 iconographic descriptions, in the following called *designs*. All designs are stored in a relational MySQL database.

The CNT designs differ a lot in level of detail and length. Some consist of merely four words:

> "Bust of Dionysus right."

Others describe a scene in detail:

> "Emperor (Caracalla) and Heracles; to left, laureate and cuirassed emperor (Caracalla) standing facing, head to right, wearing boots, holding a long sceptre in his right hand; to right, nude Herakles, standing left, holding club in left hand and showing him with right hand the apples of the Hesperides; between them, below their clasped hands, round garlanded lighted altar. Ground line. Border of dots."

Semicolons are used to indicate the beginning of a semantic unit and to structure the design by focusing on an iconographic subject. Yet it is not always obvious who the iconographic subject is. In this design, the iconographic subjects are the two persons depicted on the coin.

Some of the designs are therefore accompanied by manually extracted *key words* which are also stored in the database. In

> "Apollo seated left on omphalos, holding bow in right hand.",

the coherent key words are "Apollo" and "bow". The easiest way to employ key words is a full text search without a hierarchical structure. Manual navigation through the search mask is not required. In the CNT Advanced Search, key words can be selected from a prepared list without any hierarchical structure. These key words correspond to those stored in the database. "Apollo", for example, can be selected from a drop down menu in the search mask and this proper name corresponds to "Apollo" listed in the key words in the database.

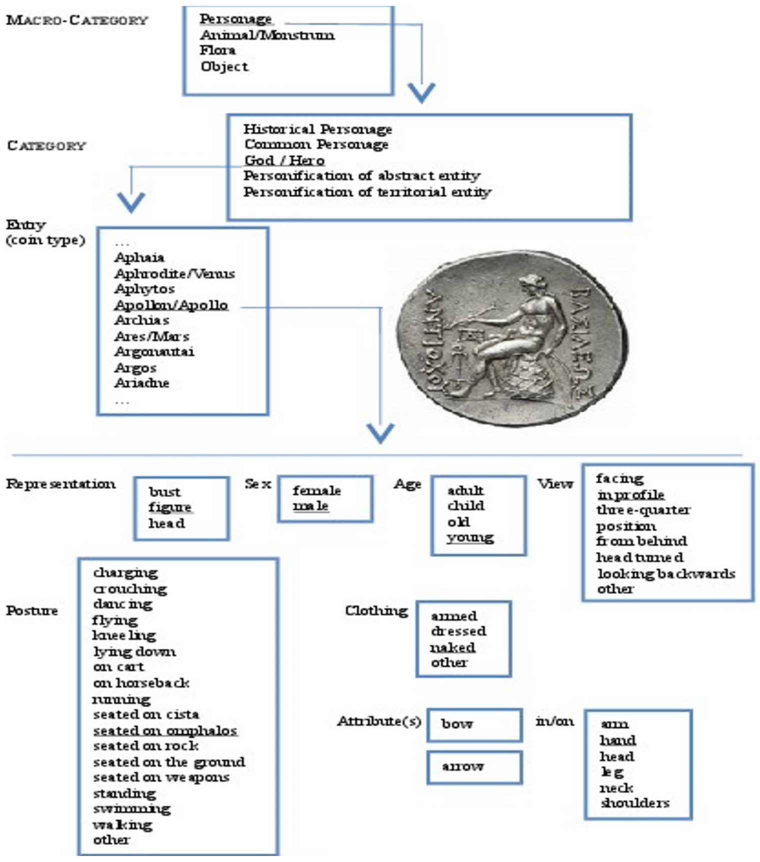


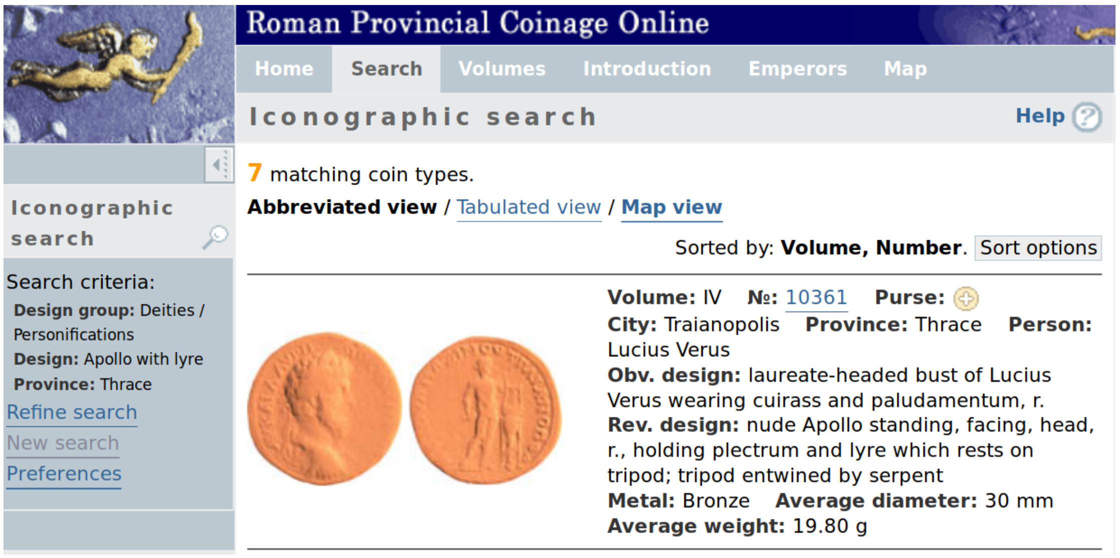
Figure 1: (Celesti et al. 2017, p. 10): The Digital Iconographic Atlas of Numismatics in Antiquity (DIANA) enables a hierarchical search. Each navigational step requires a manual click.

2.2 Related Work

A similar numismatic project is the *Digital Iconographic Atlas of Numismatics in Antiquity (DIANA)*[9]. DIANA focuses on the iconographic subject displayed on the coin and therefore distinguishes four iconography types: personages, animals/mythical creatures, flora and objects. This classification is based on numismatic indices and lexicons (Celesti et al. 2017, p. 5). DIANA enables a hierarchical search starting with the macro-category "personage" over "god/hero" to a proper name, e.g. "Apollo" (figure 1). Furthermore, attributes as the age, the posture and the clothing can be selected. For example, the posture "seated on omphalos" takes the relation of the person "Apollo" and the object "omphalos" into account (figure 1). Attributes are based on key words listed within each category. Hence each step has to be taken with a manual click to navigate in DIANA's search hierarchy.

DIANA offers a broad range of search options as well as the CNT Advanced Search. Similarly, other numismatic databases like *Mantis: A Numismatic Technologies Integration Ser-*

[9]http://ww2.unime.it/diana/


Roman Provincial Coinage Online

Home Search Volumes Introduction Emperors Map

Iconographic search Help

7 matching coin types.

Abbreviated view / Tabulated view / Map view

Sorted by: **Volume, Number**. Sort options

Volume: IV **№:** 10361 **Purse:**
City: Traianopolis **Province:** Thrace **Person:** Lucius Verus
Obv. design: laureate-headed bust of Lucius Verus wearing cuirass and paludamentum, r.
Rev. design: nude Apollo standing, facing, head, r., holding plectrum and lyre which rests on tripod; tripod entwined by serpent
Metal: Bronze **Average diameter:** 30 mm
Average weight: 19.80 g

Iconographic search

Search criteria:
Design group: Deities / Personifications
Design: Apollo with lyre
Province: Thrace

Refine search
New search
Preferences

Figure 2: In the Roman Provincial Coinage Iconographic search, both design groups and key words serve as search criteria. Here the first matching coin is displayed.

vice (MANTIS)[10], *Online Coins of the Roman Empire (OCRE)*[11] and *Roman Provincial Coinage (RPC)*[12] can be queried by specific key words. In the RPC Iconographic search[13], seven design groups are distinguished: animals, architecture, deities/personifications, games, heroes/famous persons, imperial family and objects. By selecting one group, e.g. deities/personifications, key words like “Apollo with lyre” can be selected from a prepared list without any hierarchical structure. After choosing a province from an analogously structured list, e.g. “Thrace”, all coins from this province are shown (figure 2).

Of the previously discussed search approaches, DIANA’s is the most sophisticated: It is structured hierarchically, distinguishes different iconography types and takes certain relations between these types into account. Still, if search is based on key words in the search mask or in the database, manual labor is required to select entries and to add new designs. Thus search does not improve automatically by learning new categories. Relevant information in new added designs is not recognized automatically. Given the discussed approaches, it’s fair to say that NLP has not yet been applied to numismatics. The goal is to extract relevant information from designs and enhance search in terms of automation, flexibility and generalizability.

Based on the idea introduced by DIANA to focus on the iconographic subject, this thesis takes first steps in addressing the discussed shortcomings by making use of NLP. The theoretical background needed therefor is presented in the following chapter.

[10]http://numismatics.org/search/search
[11]http://numismatics.org/ocre/search
[12]http://rpc.ashmus.ox.ac.uk/coins/
[13]http://rpc.ashmus.ox.ac.uk/search/icono/

3 Methods and Theoretical Background

This chapter provides an overview of the methods and the theoretical background relevant for this bachelor thesis. The order of this chapter follows the implemented *named entity recognition (NER)* and *relation extraction (RE)* pipelines (cf. section 4). All implementations interact with the spaCy NLP library.

In section 3.1, the information extraction architecture with its crucial steps NER and RE is discussed. In section 3.2, classification and multiclass classification are defined. The techniques to extract the most adequate features to train a classifier are discussed in section 3.3. The metrics and best practices to evaluate a classifier's performance are presented in section 3.4. In section 3.5, cross-validated grid search is introduced. spaCy's classes `DependencyParser` and `EntityRecognizer` are discussed in section 3.6. Different methods for RE are presented in section 3.7.

3.1 Information Extraction Architecture

The founders of the *Natural Language Toolkit (NLTK)* Bird, Klein and Loper have coined the notion of *information extraction* by presenting an information extraction architecture (Bird, Klein, and Loper 2009). The pipeline architecture for an information extraction system consists of five steps (figure 3). In the first step of sentence segmentation, the raw

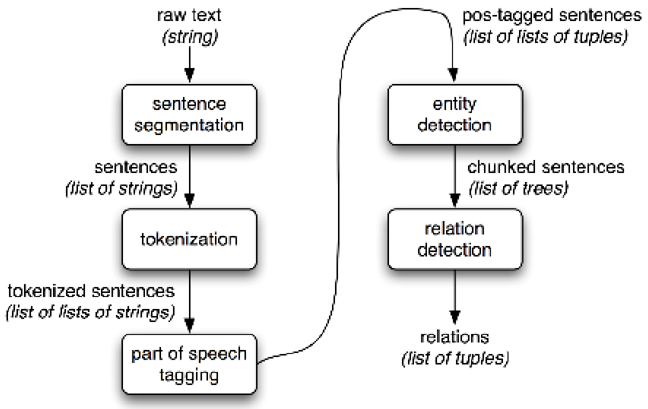


Figure 3: (Bird, Klein, and Loper 2009, chap. 7): Information Extraction Architecture. Five steps are required to transform raw text into relations.

input text is subdivided into sentences as the basic units of meaning (Bird, Klein, and Loper 2009, chap. 7.1). To access each word of these sentences represented by a list of strings, the sentences are *tokenized* in a second step so that each string is divided into substrings. Third, the tokenized sentences are *part of speech (PoS) tagged* so that each word is labeled with its grammatical functionality in the sentence, e.g. with noun, verb, adverb or determinant. In the fourth step, *named entities (NEs)* are recognized in the PoS tagged sentences. NEs are definite noun phrases, e.g. proper names, that refer to specific types of individuals, e.g. persons, organizations or locations (Bird, Klein, and Loper 2009, chap. 7.5). *Named entity recognition (NER)* shall identify all textual mentions of the NEs (Bird, Klein, and Loper 2009, chap. 7.5). The basic technique used in this context is *chunking* in order to extract the definite noun phrases in a sentence. Chunking is often realized with regular expressions (Bird, Klein, and Loper 2009, chap. 7.2). The fifth and last step in the pipeline is *relation extraction (RE).* To extract the relation between NEs means to built triples of the form (NE_1, α, NE_2) with α being "the string of words that intervenes between the two NEs" (Bird, Klein, and Loper 2009, chap. 7.6). RE can be used, for example, to reveal an existing relation indicated by the verb between a subject and an object in a sentence.

3.2 Classification

A problem is a *classification* problem if each input vector is assigned to a finite number of discrete classes (Bishop 2006, p. 3). The classification process consists of two processes: *Training* and *prediction* (Sarkar 2016, p. 171). During training, a *supervised learning* algorithm takes the annotated input data, i.e. the input data together with the class to which the coherent data point shall be assigned, and returns the trained classifier. Usually, the training data has to be annotated manually to provide the classifier the information it requires in the training process. During prediction, the trained classifier predicts the class of input vectors which have not been involved in the preceding training process.

This bachelor thesis focuses on *multiclass classification* with each input vector being assigned to one of k mutually exclusive classes (Bishop 2006, p. 235). Accordingly, each prediction can contain any of these k classes (Sarkar 2016, p. 172).

There exists a variety of classifiers. For text classification, *support vector machines (SVMs)* are an effective choice (Sarkar 2016, p. 194). SVMs choose the *decision boundary* so that the margin, i.e. the smallest distance between the decision boundary and any data sample, is maximized (Bishop 2006, p. 326). The decision boundary separates the input space into one *decision region* for each class so that all data points are assigned to one concrete class (Bishop 2006, p. 39). Other classifiers are *logistic regression* and *LSTMs* (for more details cf. section 3.6). Logistic regression is a generalized linear model based on the logistic sigmoid function and is also commonly used for classification problems (Bishop 2006, p. 205).

3.3 Feature Extraction in Natural Language Processing

To train a classifier so that it produces the optimal output, it is necessary to find adequate features as input. A *feature* is understood as a unique, measurable attribute or property for each data point in a data set (Sarkar 2016, p. 177). Thus *feature extraction* is a pre-processing stage to provide a subsequent classification algorithm with distinguishable input between different classes (Bishop 2006, p. 2). Feature extraction is "both art and science" (Sarkar 2016, p. 178). During this process, it has to be taken into account that many machine learning algorithms require a fixed-length feature vector as input (Le and Mikolov 2014). However, words and sentences come in variable length. Therefore several techniques have been developed to transform text data into fixed-length vectors. *Bag of* *words* and *Term Frequency-Inverse Document Frequency (TF-IDF)* as the most common ones are presented in the following.

In a *bag of words* representation, the frequency of each word appearing in a text is counted. Bag of words has become a commonly used model to extract features from text documents due to its simplicity and power (Sarkar 2016, p. 179). It is computed in three steps: tokenize the document, build a vocabulary over all words in the document and finally count how frequently each vocabulary word appears in the document (Muller and Guido 2017, p. 327). The output is a vector of word counts per document.

Bag of words features have two major disadvantages: The ordering of the input words is lost and the semantics of the words is ignored (Le and Mikolov 2014). Using bag of words with *n-grams*, i.e. sequences of tokens of length n, partially preserves the context of the words. Commonly, pairs (*bigrams*) or triplets (*trigrams*) of tokens that appear next to each other are used (Muller and Guido 2017, p. 339). Bag of words is implemented in the class `sklearn.feature_extraction.text.CountVectorizer`. It has the optional parameter `ngram_range` to specify the range of the sequences of tokens' length. The default value is to use single tokens (*unigrams*).

Instead of merely counting the occurrences of words, *Term Frequency-Inverse Document Frequency (TF-IDF)* highly weights any term that occurs often in a particular document (Muller and Guido 2017, p. 336). Mathematically, TF-IDF is the product of two metrics *tf* and *idf* with tf being the document frequency and idf being the inverse of the document frequency for each term (Sarkar 2016, p. 181f). To compute TF-IDF, one has to divide the total number of documents in the corpus by tf for each term and apply logarithmic scaling on the result. TF-IDF is implemented in the class `sklearn.feature_extraction.text.TfidfVectorizer` taking the text data as input and performing bag of words as well as the TF-IDF transformation on it (Muller and Guido 2017, p. 336). Like bag of words, TF-IDF has an `ngram_range` parameter.

Another word vectorization model is the *word2vec* algorithm. The word2vec model is a neural network–based model which learns distributed vector representations of words (Sarkar 2016, p. 187). Word2vec is based on a skip-gram model that learns the word vector representations which are good at predicting the nearby words (Mikolov et al. 2013, p. 3112). The word2vec model outperforms other approaches in finding the closest entities in a sentence with an accuracy of 72% (for the definition of accuracy cf. section 3.4). However, this score is only achieved with a huge amount of training data (Mikolov et al. 2013, p. 3117). Furthermore, the choice of the training algorithm and the hyper-parameter selection heavily depends on the specific task and has to be explored individually by parameter tuning (Mikolov et al. 2013, p. 3118), (for parameter tuning cf. section 3.5).

Nevertheless, it is still a problem to vectorize documents with word2vec. A simple approach is to average over all the words in the document. However, weighted averaging of word vectors loses the ordering of the input words similar to bag of words (Le and Mikolov 2014). This is not the case for ParagraphVector, an unsupervised algorithm learning fixed-length feature representations from variable-length texts (Le and Mikolov 2014). ParagraphVector is based upon the word2vec model in which every paragraph and every word is mapped onto a unique vector. The paragraph receives a paragraph token which remembers what is missing in the current context (Le and Mikolov 2014).

	p' (predicted)	n' (predicted)
p (actual)	TP	FN
n (actual)	FP	TN

Table 1: (Sarkar 2016, cf. p. 201): Confusion matrix for a two-class classification problem. TP = True Positives, FP = False Positives, FN = False Negatives and TN = True Negatives.

Le and Mikolov showed for one concrete information retrieval task how ParagraphVector minimizes the error rate up to 32% compared to bag of words, bag of bigrams and averaging word vectors. But again this score has so far only been achieved with a huge amount of training data, precisely with one million queries from a database (Le and Mikolov 2014).

3.4 Evaluation of a Classifier's Performance

After having trained a classifier, it is interesting to estimate its performance for new, previously unseen inputs. By splitting the present input data into a training and a test set, it can be estimated how well the classifier is expected to predict unseen data. Evaluation techniques analyze a classifier by comparing the labels it generates for the inputs in a test set with the correct labels for those inputs (Bird, Klein, and Loper 2009, chap. 6.3).

There are several metrics for evaluating a classifier's performance. *Accuracy*, *precision*, *recall* and *F1 score* as the most common ones are presented in the following (Sarkar 2016, p. 200).

To define these metrics, True Positives (TP), False Positives (FP), False Negatives (FN) and True Negatives (TN) are introduced (table 1). "p" denotes the positive and "n" the negative class. TPs are defined as the number of instances correctly predicted and belonging to the positive class, whereas FNs are the number of instances falsely predicted and belonging to the negative class. Accordingly, FPs are the number of instances wrongly predicted and belonging to the positive class, whereas TNs are the number of instances correctly predicted and belonging to the negative class (Sarkar 2016, p. 201).

The simplest metric that can be used for evaluation is *accuracy.* Accuracy measures the percentage of annotated input data that has been labeled correctly by the classifier (Bird, Klein, and Loper 2009, chap. 6.3). It is understood as the overall proportion of correct predictions of the classifier (Sarkar 2016, p. 202):

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

A different measurement approach offer the metrics *precision* and *recall.* Precision indicates how many of the identified items are relevant (Bird, Klein, and Loper 2009, chap. 6.3). It is defined as the fraction of the correct predictions made by the classifier out of all predictions (Sarkar 2016, p. 203):

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recall indicates how many of the relevant items were identified (Bird, Klein, and Loper 2009, chap. 6.3). It is defined as the fraction of correctly predicted instances (Sarkar 2016, p. 203):

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

There is a trade-off between precision and recall: If one metric is optimized, the other has to be slightly neglected (Muller and Guido 2017, p. 282f). Therefore, the *F1 score*, i.e. the harmonic mean of precision and recall, can be used as an alternative metric:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{4}$$

It strongly depends on the application's necessities which evaluation metric to use.

3.5 Cross-Validated Grid Search

To improve a classifier's performance, its parameters can be tuned. For parameter tuning, *grid search* with *cross-validation* is a common choice (Muller and Guido 2017, p. 265). In a *grid search*, different parameter combinations are probed over a grid. The performance of each parameter combination is evaluated by *cross-validation* (Pedregosa et al. 2011). For cross-validation, the data set is split into S equally sized folds. A proportion $(S-1)/S$ is used for training and the remaining proportion $1/S$ for testing. This procedure is repeated for all S possible choices (Bishop 2006, p. 32f). Cross-validated grid search is implemented in the class `sklearn.model_selection.GridSearchCV`.

3.6 SpaCy's DependencyParser and EntityRecognizer

The implementations of NER and RE in this bachelor thesis interact with the spaCy NLP library. In the following, spaCy's classes `DependencyParser` and `EntityRecognizer` are discussed.

`DependencyParser` annotates syntactic dependencies on natural language texts. According to spaCy's founder Matthew Honnibal, the class is based on a transition-based parsing system[14]. In such system, the parser consists of a state which can be manipulated by a set of actions (Honnibal, Goldberg, and Johnson 2013, p. 164). A particular set of states produces a transition-system.

Honnibal focuses on the monotonic arc-eager transition system in which four parsing actions are distinguished: *Shift*, *Right-Arc*, *Left-Arc* and *Reduce*. *Shift* pops the first item on the buffer and pushes it on the stack. *Right-Arc* additionally adds a dependency arc from the top of the stack to the first item on the buffer. *Left-Arc* adds a dependency arc from the first item on the buffer to the top of the stack. Finally, *reduce* pops all items from the top of the stack. The output is supposed to be a well-formed parse tree. Given the monotonicity of the system, "once an action has been performed, subsequent actions must be consistent with it."(Honnibal, Goldberg, and Johnson 2013, p. 164) If a wrong decision is made, the parser cannot recover from an incorrect head assignment. In

[14]https://explosion.ai/blog/how-spacy-works

a non-monotonic arc-eager system, however, a previous decision can be undone because a subsequent move can overwrite an attachment made by the first (Honnibal, Goldberg, and Johnson 2013, p. 169).

`EntityRecognizer` annotates NEs on natural language texts. The foundations[15] of this class are the previously discussed transition-based dependency parsing and *Long short-term memory (LSTM)*, a type of recurrent neural network. In the class `EntityRecognizer`, these two concepts are combined in a *stack LSTM* to incrementally construct chunks of the input (Lample et al. 2016). According to Lample, LSTMs are a reasonable choice for NLP tasks because in natural language, important information is position dependent. Therefore LSTMs are a better choice for modeling the relationship between words and their characters than, for example, convolutional networks, although LSTMs have a representation biased towards their most recent inputs (Lample et al. 2016).

In a stack LSTM, the input layers are vector representations of individual words. To get this input, pretrained word embeddings, precisely skip-n-grams as a variation of word2vec, have been used (cf. section 3.3). In the chunking algorithm, transition-based dependency parsing is employed. For presenting labeled chunks, a bidirectional LSTM is used. The output is a single vector representation for each labeled, generated chunk, whereas the length of the chunk is irrelevant (Lample et al. 2016). All things considered, stack LSTMs outperform other approaches in sequence labeling, even those which use external resources (Lample et al. 2016).

3.7 Methods for Relation Extraction

The goal of RE is to extract relations between NEs. The main question can be framed as: What relations exist between two NEs? In fact, RE targets a very difficult problem: A pair of NEs can have more than only one relation (Konstantinova 2014, p. 22) and language ambiguity is a major obstacle (Konstantinova 2014, p. 24). Currently, the methods used for RE can be differentiated into three categories: *knowledge-based methods*, *supervised methods* and *self-supervised methods* (Konstantinova 2014, p. 17). Due to its complexity, the latter was not taken into consideration for this bachelor thesis.

Knowledge-based methods are used for domain-specific tasks in which a fixed set of relations has to be extracted. Usually these methods are based on pattern-matching rules. Annotations and training are not necessary. However, knowledge-based methods usually do not generalize to other data sets and demand much manual labor (Konstantinova 2014, p. 18).

Supervised methods automatically learn how to extract relations and therefore easily adapt to other data sets. Supervised relation extraction can be performed with kernel methods, logistic regression or conditional random fields (Konstantinova 2014, p. 18). The drawback of supervised methods is that a large amount of training data is needed and that it might take a lot of manual labor to create the annotated corpus required to train the model (Konstantinova 2014, p. 18).

All things considered, it has to be dealt with a trade-off between manual labor

[15] https://spacy.io/usage/resources#videos

for creating appropriate training data and having a system that learns automatically when performing RE on a data set. Knowledge-based methods optimize for the former, supervised methods for the latter.

Once a decision for a method has been made, another challenge is to find adequate features (cf. section 3.3). Different approaches to RE have shown two interesting features. First, if focusing on relation-independent lexico-syntactic patterns, it is promising to focus on the verb, because "95% of all relations in English can be described by only eight general patterns" (Konstantinova 2014, p. 20), such as triples of the form $(NE_1, verb, NE_2)$ (cf. section 3.1). A second promising feature is to use the dependency parser (cf. section 3.6) and extract the path in the dependency parse tree between two NEs because in English, the relations are often found in between two NEs (Konstantinova 2014, p. 21).

4 Implementation

In this bachelor thesis, *named entity recognition (NER)* and *relation extraction (RE)* are implemented in two sequential pipelines. In accordance with the information extraction architecture, NER is the step before RE (cf. section 3.1). The initial input for the NER pipeline is a single iconographic description (*design*) from the CNT database (figure 4). After passing the NER pipeline, the *named entities (NEs)* in the design are assigned to one of four labels: PERSON, ANIMAL, PLANT or OBJECT. In this bachelor thesis, only the PERSONs, i.e. the subjects, and the OBJECTs, i.e. the objects, are relevant for the RE pipeline. This pipeline uses the design together with the NEs labeled as PERSONs and OBJECTs as input to predict relations between subjects and objects. Therefore the output of the NER pipeline is transformed into a list of (design, subject, object)-triples as input for the RE pipeline. In this step, the cross product of all NEs labeled as PERSONs or OBJECTs is taken and only the (PERSON, OBJECT) pairs are considered. After passing the RE pipeline, one or multiple (subject, relation, object)-triples are extracted from the initial input design.

For a better understanding of the data flow, consider the following example design:

> "Apollo seated left on omphalos, holding bow in right hand."

After passing the NER pipeline, "Apollo" is labeled as PERSON and "omphalos" and "bow" are each labeled as OBJECT:

> " Apollo PERSON seated left on omphalos OBJECT , holding bow OBJECT in right hand."

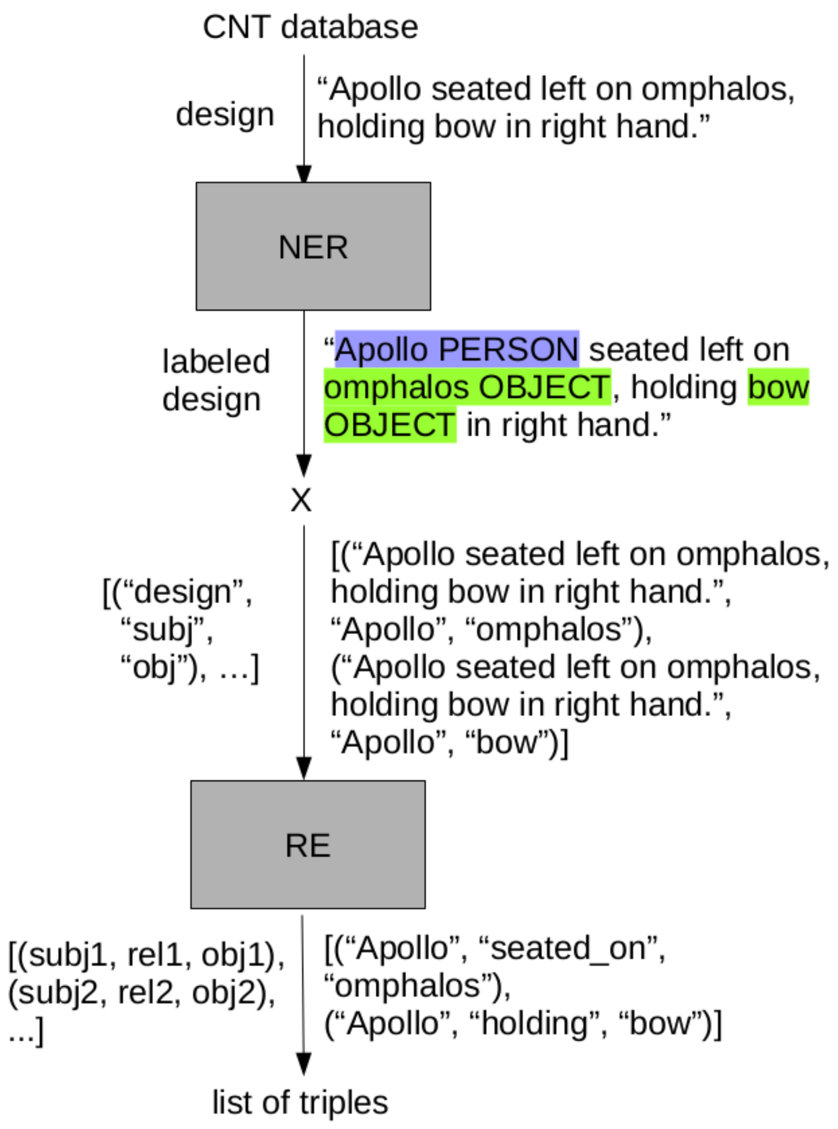

CNT database
design
“Apollo seated left on omphalos, holding bow in right hand.”
NER
labeled design
“Apollo PERSON seated left on omphalos OBJECT, holding bow OBJECT in right hand.”
X
[(“design”, “subj”, “obj”), …]
[(“Apollo seated left on omphalos, holding bow in right hand.”, “Apollo”, “omphalos”), (“Apollo seated left on omphalos, holding bow in right hand.”, “Apollo”, “bow”)]
RE
[(subj1, rel1, obj1), (subj2, rel2, obj2), ...]
[(“Apollo”, “seated_on”, “omphalos”), (“Apollo”, “holding”, “bow”)]
list of triples

Figure 4: Named Entity Recognition and Relation Extraction are implemented in two sequential pipelines.

Each (PERSON,OBJECT) pair is presented as a candidate for the following RE pipeline:

[(“Apollo seated left on omphalos, holding bow in right hand.”,
“Apollo”, “omphalos”),
(“Apollo seated left on omphalos, holding bow in right hand.”,
“Apollo”, “bow”)].

After passing the RE pipeline, the output ideally looks like this:

[(“Apollo”, “seated_on”, “omphalos”), (“Apollo”, “holding”, “bow”)].

“seated_on” and “holding” are the relations that exist between the subject “Apollo” and the objects “omphalos” and “bow”.

After this high-level overview, the concept and workflow of the two inner pipelines are explained in more details in section 4.1 and section 4.2.

4.1 Named Entity Recognition

To identify the different NE types in the CNT designs, NER is performed by employing spaCy’s class `EntityRecognizer` (cf. section 3.6). The first implementation step was to perform NER on persons only as proof of principle. Therefore a list of persons was created manually by using the key words given in the relational database and by screening the English designs for synonyms, homonyms, alternative spellings and spelling errors of these persons’ names. This list serves as input for the NER workflow (figure 5). In step **1**,

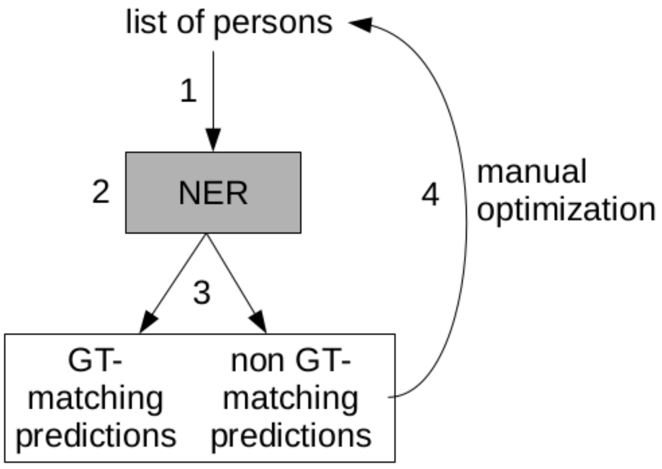


Figure 5: In the Named Entity Recognition workflow, revising the ground truth (GT)-matching and non ground truth (GT)-matching predictions helps to improve the input for the Named Entity recognizer.

the NE recognizer gets the list of persons which is used for annotating the CNT designs automatically for the training process. Each automatically generated annotation consists of a list of (start, end, label)-triples where start denotes the position of the first character in the string of the NE, end denotes the position of the last character and label is the corresponding label. For

"Apollo seated left on omphalos."

the corresponding annotations are

[(0, 6, PERSON), (22, 30, OBJECT)].

Before training, the data has to be split into a train and a test set (cf. section 3.4) to evaluate the NE recognizer's performance. Two different types of train-test splits have been employed: a random split implemented with `sklearn.model_selection.train-_test_split` and a name-disjoint split yielding disjoint sets of persons in the training and in the test set. In the second case, the NE recognizer receives test data that contains names of persons unseen in the training process. The idea of this name-disjoint split is to investigate if the NE recognizer is able to predict the correct label for new, unseen names of persons.

In step **2** of the NER workflow, the NE recognizer is trained on the train set with three pipeline iterations (figure 5). Afterwards, the trained NE recognizer receives the unseen, unlabeled test set and predicts the corresponding labels for the recognized NEs (cf. section 3.2).

After training, the NE recognizer's output is distinguished in ground truth (GT)-matching and non ground truth (GT)-matching predictions in step **3** (figure 5). Predictions match the ground truth if the predicted labels equal the ground truth annotations and are non GT-matching otherwise. In step **4**, the list of persons can be improved by reviewing the predictions manually (figure 5).

By looping over the NER workflow iteratively, the list of persons was successively improved. The evaluation of this optimization process is discussed in section 6.1. Once the performance of the NE recognizer was satisfying, additional lists including objects, animals and plants were created. Together with the list of persons they were used as input for the NE recognizer which then was trained on the four labels PERSON, OBJECT, ANIMAL and PLANT in order to predict those labels for unseen data. Again, the so extended NER workflow was improved in iterative loops (figure 5).

The NE recognizer's performance is evaluated by using the metrics accuracy, precision and recall (cf. section 3.4). As spaCy does not offer a customized function to evaluate accuracy, an accuracy function has been implemented. Accuracy is defined as the fraction of entirely correctly predicted designs out of all n predictions:

$$Accuracy = \frac{1}{n}\sum_{i=1}^{n}\delta(y_{\text{true},i}, y_{\text{pred},i}) \tag{5}$$

with

$$\delta(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Accuracy is an adequate metric to measure the NE recognizer's performance in the NER pipeline. However, it is a tough criterion, as a prediction is labeled as correct if and only if all NEs are labeled correctly. If, for example, a design contains five NEs and only one receives a wrong label or is not labeled at all, the design is labeled incorrectly in total. Precision and recall, however, are defined for each NE individually. Therefore these metrics are additionally used for evaluation (cf. section 3.4).

Scikit learn offers the functions `sklearn.metrics.recall_score(y_true, y_pred)` and `sklearn.metrics.precision_score(y_true, y_pred)`. However, these functions are restricted to the binary classification problem (Pedregosa et al. 2011). For the NE recognizer, a precision-recall function has been implemented. Precision is defined as the cardinality of the intersection of the set of predicted NEs and the set of ground truth NEs divided by the cardinality of the set of predicted NEs:

$$Precision = \frac{|prediction \cap groundtruth|}{|prediction|} \tag{7}$$

Recall is defined as the cardinality of the intersection of the set of predicted NEs and the set of ground truth NEs divided by the cardinality of the set of ground truth NEs:

$$Recall = \frac{|prediction \cap groundtruth|}{|groundtruth|} \tag{8}$$

4.2 Relation Extraction

RE is the task of extracting relations between NEs and a challenging trade-off is the one between manual labor and automatically learning systems (cf. section 3.7). Given the CNT data with 3.628 designs and its relations not being annotated, on the one hand the data set is too small for a supervised learning approach and it is impracticable to manually annotate the whole data set. On the other hand, a knowledge-based approach using patterns and rules does not generalize. So how can relations be extracted from the CNT designs?

By rephrasing the problem, this bachelor thesis proposes a different approach to RE. If the types of relations in the data set is known, instead of asking what relations exist between a pair of NEs, it can be asked if a pair of NEs has a certain relation or not. This way, the problem can be understood as a multiclass classification problem which can be solved by employing classifiers like logistic regression or SVMs (cf. section 3.2). This has the advantage that creating training data is comparatively easy because only the relations have to be annotated. However, new relations cannot be learned as the number of relations is as fixed as the labels in the NER pipeline.

To produce a small, but functional application, the focus in this bachelor thesis is exclusively on (NE_1, α, NE_2)-triples with NE_1 being the iconographic subject on the coin labeled as PERSON, α being a relation represented by a verb and with NE_2 being the object on the coin labeled as OBJECT. Thus the considered designs are only those containing at least one PERSON and one OBJECT label.

Each design in the CNT data has been annotated manually analogous to the output of the NER pipeline with a list of (PERSON, relation, OBJECT)-triples if a desired relation existed between them (cf. figure 4). The relations in

relation	semantic cluster
holding	“holding”, “carrying” (garment), “brandishing” (spear), “shouldering” (rudder), “playing” (lyre, aulos), “raising” (shield), “cradling” (torch)
wearing	“wearing”, “covered with” (lion-skin)
resting_on	“resting on”, “leaning on”
seated_on	“seated on”, “seated in”
standing	“standing in” (biga), “driving” (biga), “standing on” (galley)
drawing	“drawing” (arrow)
stepping_on	“stepping on” (helmet)
grasping	“scooping” (gold), “reach out for” (person), “plucking” (chiton)
lying	“lying on”
hurling	“hurling” (thunderbolt)
no_existing_relation	

Table 2: The eleven relations identified between (PERSON, OBJECT) pairs in the CNT designs represent semantic clusters. no_existing_relation is used for designs which lack any such relation.

> “Apollo seated left on omphalos, holding bow in right hand.”

are seated_on and holding, thus the corresponding annotations are

> [(“Apollo”, “seated_on”, “omphalos”), (“Apollo”, “holding”, “bow”)].

In total, eleven relations were identified in the CNT data set during the annotation process. Each relation class comprises a number of semantically similar relations (table 2).

In some cases, it was useful to annotate more relations than only those explicitly represented by verbs. For

> “Nike in biga, right.”

the corresponding annotation is

> [(“Nike”, “standing”, “biga”)]

because “standing” is the relation used in all other similar designs depicting Nike in a biga and can thus be annotated although not being explicitly mentioned. Designs which do not contain any of the desired relations are annotated as a “no_existing_relation”. For

> “Bust of Dionysus right.”

the corresponding annotation is an empty list [ ].

The CNT designs together with the PERSONs and OBJECTs detected in the NER pipeline serve as input for the RE workflow (figure 6). In step **1**, the (design, subject,

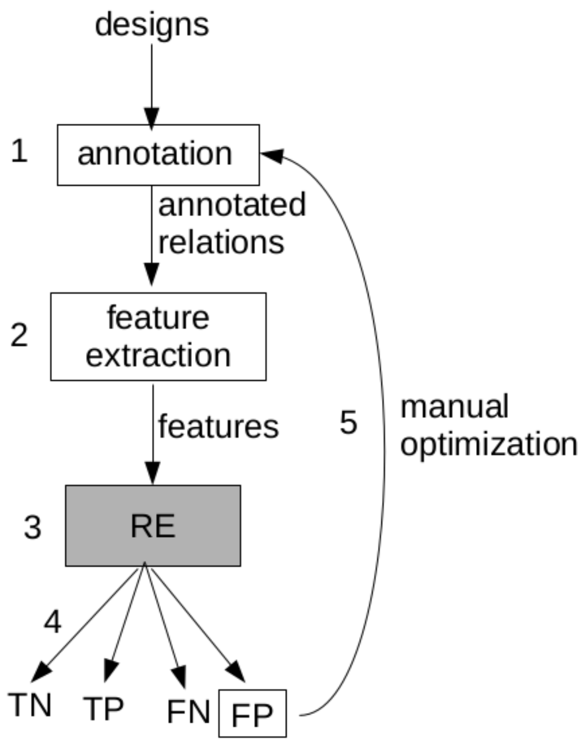


Figure 6: In the Relation Extraction workflow, annotated relations are transformed into features for training the Relation Extraction classifiers. TN = True Negatives, TP = True Positives, FN = False Negatives and FP = False Positives.

object)-triples are annotated manually with relations between (PERSON, OBJECT) pairs. In step **2** of the RE workflow, features are extracted from the annotated relations (figure 6). These features serve as input for the training process because before training a classifier, adequate features have to be found (cf. section 3.3). To use a vectorizer, the designs have to be preprocessed. Hence preprocessing features and vectorizers have to be combined. Word vectorization models like word2vec and doc2vec, on the contrary, do not need an additional vectorizer (cf. section 3.3). In the following, the selected **features** are presented:

Doc2Str is a simple feature that applies a vectorizer such as TF-IDF to the whole design (cf. section 3.3). This feature does not use the NE labels in the designs and is therefore expected to perform poorly.

Path2Str To create a better feature, it seemed promising to extract the path on the parse tree between two NEs (cf. section 3.7). The path is extracted in a path function which determines the least common ancestor of the two nodes subject and object and returns the whole path between them if existing. `Path2Str` is augmented with two syntactic-based features as optional parameters: part of speech (PoS)-tags (cf. section 3.1) and dependency labels which show the syntactic dependencies between words in a sentence. Both are accessible with spaCy's `DependencyParser` after having generated the parse tree (cf. section 3.6). The parse tree including PoS-tags and dependency labels can be depicted with the visualizer displaCy[16].

Verb2Str uses the PoS-tag "verb" on the path of the parse tree between two NEs. Whenever the PoS-tag of a word is a verb, it is added to the feature.

AveragedPath2Vec implements word2vec and additionaly averages over all words on the path (cf. section 3.3). It employs spaCy's `vector` class[17].

AveragedRest2Vec is the complementary input to word2vec. Not the path of the parse tree between two NEs is extracted, but the rest of the design except for the path.

Doc2Vec implements the word vectorization model doc2vec by interacting with the `gensim` library[18] (cf. section 3.3).

In step **3** of the RE workflow, the extracted features are used as input for training the RE classifiers (figure 6). Three different classifiers were probed: SVMs both with a linear and a radial basis function (rbf) kernel implemented with `sklearn.svm.SVC` and logistic regression implemented with `sklearn.linear_model.LogisticRegression` (cf. section 3.2). These classifiers were trained with all possible feature combinations (for evaluation see section 6.2).

To find the best feature-classifier-combination, a cross-validated grid search with five repetitions was performed and the classifiers' performances were evaluated (cf. section 3.5). The most adequate metric to evaluate the classifiers' performances seems to be the F1 score (cf. section 3.4). Accuracy is not very adequate because a prediction of a design is labeled as correct if and only if all relations are labeled correctly. If, for example, a design contains five relations and only one receives a wrong label, the design is labeled incorrectly in total. Therefore precision and recall, and hence F1 score as the harmonic mean of the two, offer a more substantial metric for the RE pipeline (cf. section 3.4).

After having determined the best feature-classifier-combination, the classifier's output is split in True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN) in step **4** of the RE workflow (figure 6). The goal of this distinction is to analyze the FPs for further optimization.

True Positives are those predicted relations which are present in the ground truth annotations:

$$TP = prediction \cap groundtruth \tag{9}$$

[16]https://demos.explosion.ai/displacy/
[17]https://spacy.io/api/vectors
[18]https://radimrehurek.com/gensim/models/doc2vec.html

False Positives are those predicted relations which are not present in the ground truth annotations:

$$FP = prediction \setminus groundtruth \tag{10}$$

False Negatives are those relations present in the ground truth annotations, but not predicted:

$$FN = groundtruth \setminus prediction \tag{11}$$

True Negatives (TN) are those relations which are not predicted and not present in the ground truth annotations. As TNs can only be detected manually, they were not in focus.

In step **5** of the RE workflow, the annotated relations are optimized (figure 6). By analyzing the FPs, erroneous or missing ground truth annotations have been corrected manually. The annotated relations and hence the classifier's performance were successively improved by looping over the RE workflow iteratively.

5 Lessons Learned

After having discussed the concepts and the workflow of the *named entity recognition (NER)* and *relation extraction (RE)* pipelines in the previous chapter, this chapter focuses on the problems that arouse during implementation. Solutions to these problems are presented and the costs of these solutions in terms of computation time and manual labor are estimated.

Preprocessing Two problems were solved by preprocessing the CNT designs. The first problem was abbreviating "right" with "r." and "left" with "l.". These abbreviations are difficult because the period is usually interpreted as the end of a sentence by NLP tools. Therefore the abbreviations were eliminated. For example,

> "Apollo seated l. on omphalos, holding bow in r. hand."

became after preprocessing

> "Apollo seated left on omphalos, holding bow in right hand."

The second issue were upper and lower cases in the designs. If the NE recognizer in the NER pipeline was trained on words with lower cases only, these words were not recognized when written capitalized. Therefore both lower and upper case representations of objects, animals and plants were provided as input to the NE recognizer. For persons, the upper case representation was sufficient.

Annotations for NER pipeline To train the NE recognizer in the NER pipeline, the designs had to be annotated. Therefore all names in the designs were compiled in a list of persons (cf. section 4.1). This list was created manually by extracting the persons' names from the key words stored in the relational database and by screening the English designs for synonyms, homonyms, alternative spellings and spelling errors of these names. The annotation process corresponds to step **1** in the NER workflow (figure 7). At a later stage,

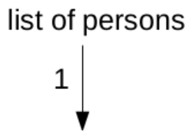


Figure 7: First step of the Named Entity Recognition workflow depicted in figure 5.

additional lists of objects, animals and plants were compiled. Words such as “centaur” required individual decisions because they could have been labeled with both PERSON and ANIMAL. Compiling all four lists took about two days of work, i.e. 16 hours of manual labor in total.

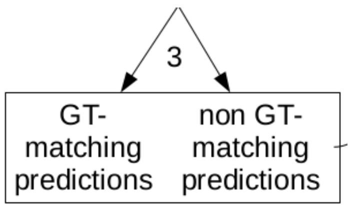


Figure 8: Third step of the Named Entity Recognition workflow depicted in figure 5.

Train-Test Split After the first run of the NER pipeline with the list of persons as input, random new names like “Joe Otto” were not correctly labeled. This was problematic because the application was supposed to generalize to other data sets. To investigate if the NE recognizer learns to predict new names, a name-disjoint train-test split yielding disjoint sets of persons was implemented (cf. section 4.1). After the name-disjoint split, the NE recognizer receives test data that contains names of persons unseen in the training process. The problem corresponds to step **3** in the NER workflow because by considering the ground truth (GT)-matching and non ground truth (GT)-matching predictions, the necessity for a name-disjoint split became clear (figure 8).

Relation Extraction A major challenge of this bachelor thesis was to figure out how relations could be extracted at all from the relatively small data set of CNT designs. This issue was coped with by rephrasing the problem and treating RE as a multiclass classification problem (for further discussion cf. section 4.2).

Annotations for RE pipeline One advantage of understanding RE as a multiclass classification problem was that the designs had to be only annotated with relations (cf. section 4.2). This approach implied two problems concerning step **1** in the RE workflow (figure 9). First, it was unclear which relations existed between (PERSON, OBJECT) pairs. Second, the annotation process had to be handled efficiently. The former issue was

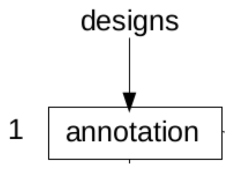


Figure 9: First step in the Relation Extraction workflow depicted in figure 6.

solved during the annotation process itself. By clustering semantically similar relations, eleven relations were identified (cf. section 4.2). The latter issue was solved by making use of the PERSONs and OBJECTs recognized in the NER pipeline. This way only the relations had to be annotated. For

“Apollo seated left on omphalos, holding bow in right hand.”,

the annotation schema looked like this:

[(“Apollo”, “ ”, “omphalos”), (“Apollo”, “ ”, “bow”)].

To identify and annotate all eleven relations existing between (PERSON, OBJECT) pairs in the CNT designs took about four days of work, i.e. 32 hours of manual labor in total (cf. table 2).

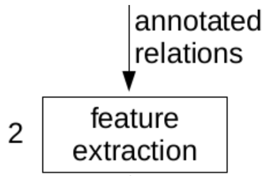


Figure 10: Second step in the Relation Extraction workflow depicted in figure 6.

Feature extraction As feature extraction is “both art and science” (Sarkar 2016, p. 178), it was challenging to find adequate features for RE. The solution was a trial and error approach: All features that seemed promising and as many as possible were tested (cf. section 4.2). The problem corresponds to step **2** in the RE workflow (figure 10).

Best Classifier-Feature Combination Given the broad variety of classifier-feature-combinations, the best combination had to be found. Therefore the different parameter combinations were probed in a cross-validated grid search with five iterations (cf. section 4.2). The cost of computation time aggregates to about five hours for one grid search because each iteration took approximately an hour on an Intel Core i3-2350M CPU with 2.30 GHz x 4.

6 Measurement Results and Discussion

6.1 Named Entity Recognition

To train the NE recognizer in the NER pipeline, the CNT designs have been annotated by using lists of persons, objects, animals and plants. The four labels differ substantially in size (figure 11): While there are almost as many ANIMALs (1.379) as PLANTs (1.405), the label PERSON is used roughly twice as often (2.844 times). The label OBJECT makes the largest difference: It occurs 7.201 times in the designs (figure 11). In comparison, there exist 166 unique OBJECTs, 163 unique PERSONs, 54 unique ANIMALs and 28 unique PLANTs in the lists. These unique terms occur with different frequencies (figure 11). The most frequent OBJECTs are bust (449 times), patera and cuirass (329 times each). The most frequent PERSONs are Apollo (192 times), Athena (171 times) and Nike (137 times). The most frequent PLANTs are branch (184 times), grain (170 times) and ears (132 times). Last but not least, the most frequent ANIMALs are lion (193 times), serpent (166 times) and horse (87 times).

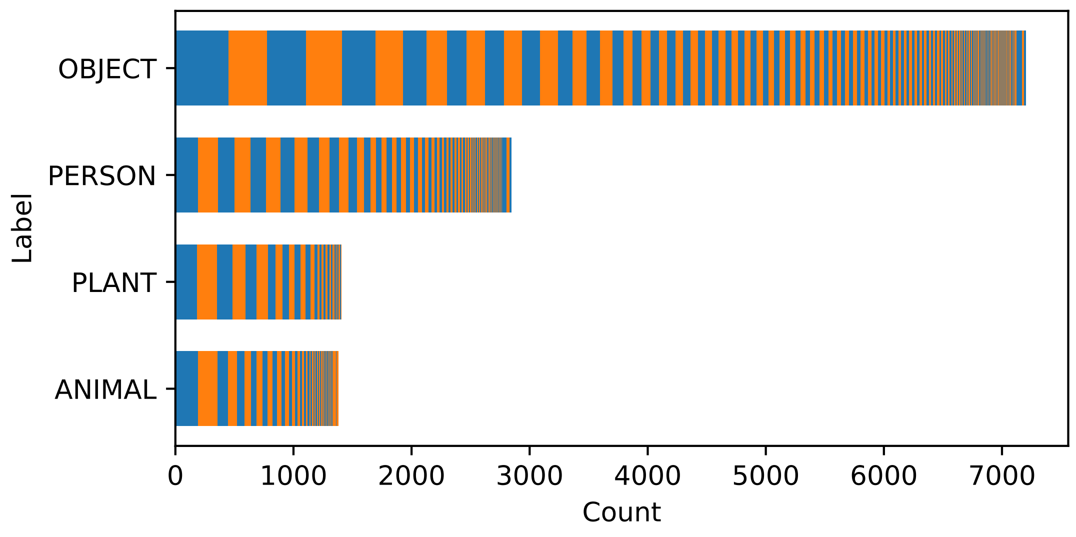


Figure 11: The number of OBJECTs is substantially higher than for any other label. Each color block corresponds to a unique term which occurs with different frequencies. For example, the most frequent PERSON is Apollo.

One reason for the large amount of OBJECTs might be that numismatic researchers are usually interested in what persons are depicted with which objects because objects often serve as attributes to identify the persons. The person Poseidon, for example, is usually shown with the object trident. Moreover, the number of persons who are depicted on an ancient coin is rather small.

Before training the NE recognizer, a random train-test split and a name-disjoint train-test split yielding disjoint sets of persons have been performed (cf. section 4.1). To

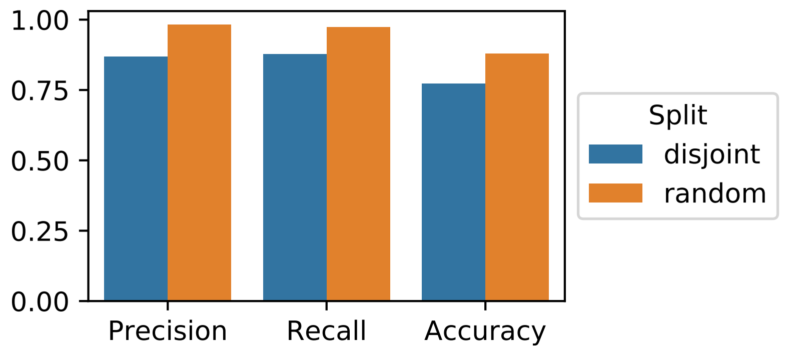


Figure 12: The Named Entity recognizer performs better with a random than with a name-disjoint train-test split on the test data.

evaluate the NE recognizer's performance after training, the metrics accuracy, precision and recall were employed. For all three metrics, the scores are better for the random split than for the name-disjoint split (figure 12): The random split achieves an accuracy of 88% on the test data, a recall of 97% and a precision of 98%, whereas the name-disjoint split achieves an accuracy of 77% on the test data, a recall of 88% and a precision of 89% (figure 12). The size of the test set was 25% of the whole data set. One reason for the better performance of the random split might be that the name-disjoint split is more demanding because the NE recognizer has to correctly predict the label of unseen person names. In the case of the random split both the train and the test set may, for example, contain "Apollo" and hence predicting the label is easier for a name already seen in the training data. All things considered, the NE recognizer performs very well, even with the more challenging name-disjoint train-test split. Nevertheless, the NE recognizer's output was reviewed after the training process to improve the ground truth annotations.

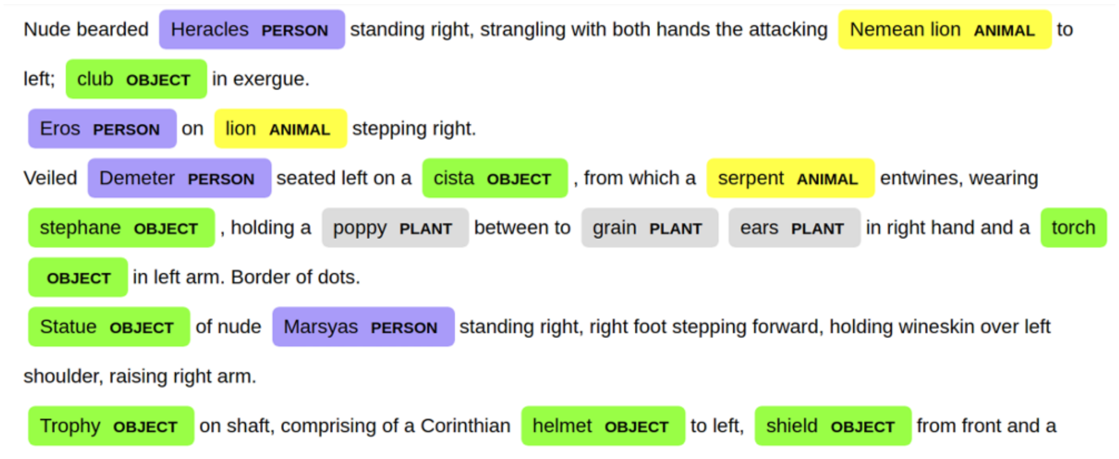
Therefore spaCy's visualizer displaCy was used to visualize ground truth-matching and non ground truth-matching predictions (figure 13). By analyzing the predictions, new persons and objects were found and added manually to the according list. For example, groups of persons like "nymphes" or "satyrs" were recognized, although not being listed in the list of persons used to automatically generate the ground truth annotations. This way the NE recognizer's performance was improved iteratively.

However, there exist also examples of missing predictions. In

> " Statue OBJECT of nude Marsyas PERSON standing right, right foot stepping forward, holding wineskin over left shoulder, raising right arm.",

"statue" is correctly labeled as OBJECT, but "wineskin" is not recognized (figure 13, design 4). If "wineskin" was a True Negative, it would not be predicted and not be present in the ground truth annotations (cf. section 3.4). This error could be eliminated by adding the word to the list of objects manually. Another mistake occurred in the following design:

> " Trophy OBJECT on shaft, comprising on a Corinthian helmet OBJECT to left, shield OBJECT from front [...] ."



Nude bearded Heracles PERSON standing right, strangling with both hands the attacking Nemean lion ANIMAL to left; club OBJECT in exergue.

Eros PERSON on lion ANIMAL stepping right.

Veiled Demeter PERSON seated left on a cista OBJECT , from which a serpent ANIMAL entwines, wearing stephane OBJECT , holding a poppy PLANT between to grain PLANT ears PLANT in right hand and a torch OBJECT in left arm. Border of dots.

Statue OBJECT of nude Marsyas PERSON standing right, right foot stepping forward, holding wineskin over left shoulder, raising right arm.

Trophy OBJECT on shaft, comprising of a Corinthian helmet OBJECT to left, shield OBJECT from front and a

Figure 13: Designs labeled in accordance with the ground truth annotations are visualized with displaCy. Each color in the visualization corresponds to a label: Purple represents PERSON, green represents OBJECT, yellow represents ANIMAL and grey represents PLANT.

"Trophy", "helmet" and "shield" are correctly labeled as OBJECTs, but "shaft" is not recognized (cf. figure 13, design 5). If "shaft" was a False Negative, it would be present in the ground truth annotations but not predicted (cf. section 3.4). This corresponds to an error of the NE recognizer. By additional loops over the NER workflow, True Negatives could be detected manually to achieve further performance improvements (cf. figure 5).

To probe generalizability, the NER pipeline was run on two external data sets from *Online Coins of the Roman Empire (OCRE)* and *Coinage of the Roman Republic Online (CRRO)*[19] based on the annotations of the CNT data with 3.628 designs, with a random train-test split and without any further adaptations. On the CRRO data set with 1.691 designs, an accuracy of 75%, a recall of 89% and a precision of 96% was achieved. On the OCRE data set with around 14.000 designs, an accuracy of 90%, a recall of 96% and a precision of 99% was achieved. These results have been achieved without revising the True Negatives and without any additional iterations over the NER workflow (cf. figure 5). Therewith it is shown that the NER implementation generalizes to other data sets and seems to improve if being used with larger data sets.

6.2 Relation Extraction

To train the classifiers in the RE pipeline, the CNT designs have been annotated with (PERSON, relation, OBJECT)-triples (cf. section 4.2). After taking the cross product of all NEs labeled as PERSONs or OBJECTs, the number of (PERSON, OBJECT) pairs is 7.589 in total. The label "no_existing_relation" is used if there doesn't exist a relation between a (PERSON, OBJECT) pair. This label occurs 3.628 times (figure 14). It is the most frequent relation because of the cross product which takes (PERSON, OBJECT)

[19]http://numismatics.org/crro/

pairs not having any relation into account. An example of a “no_existing_relation” is:

> “Bust of Dionysus right.”

In total, eleven relations have been identified in the annotation process. The other ten relations occur with different frequencies (figure 14): The second most common relations

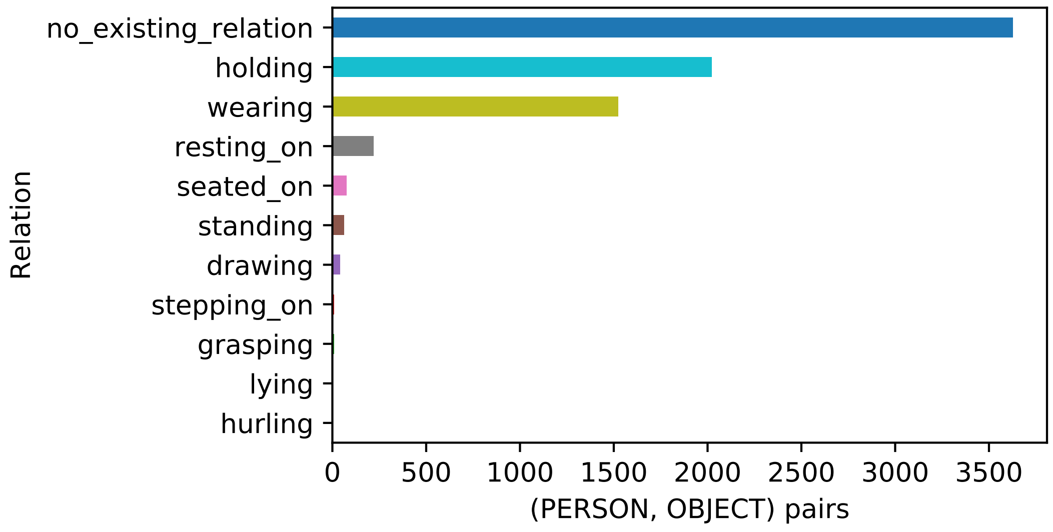


Figure 14: Eleven relations have been identified in the CNT designs. Most (PERSON, OBJECT) pairs have no relation at all. “stepping_on” and “grasping” appear eight times each, “lying” and “hurling” once each.

are “holding” appearing 2.022 times and “wearing” appearing 1.523 times. In a vast distance follows “resting_on” appearing 219 times. Less than a hundred but more than ten times appear “seated_on” (75 times), “standing” (63 times) and “drawing” (41 times). Less than ten times and therefore almost negligible are “stepping_on” and “grasping” (8 times each) as well as “lying” and “hurling” (once each). The high frequency of “holding” and “wearing” is a property of the designs: Usually persons depicted on an ancient coin are described by what they wear and what the hold, hence these are common relations between persons and objects in numismatics.

These annotations were required for training the three classifiers in the RE pipeline. Before training, a random train-test split was performed. Then all classifiers were trained with all feature combinations (cf. section 4.2). By performing a cross-validated grid search after training, the best feature-classifier-combinations were identified (cf. section 3.5). The grid search was repeated five times with each repetition taking approximately one hour to run. In total, one cross-validated grid search took about five hours time on an Intel Core i3-2350M CPU with 2.30 GHz x 4.

The top five performers with an F1 score of 88% employ a combination of the classifiers SVM with linear kernel (SVC(“linear”)) and logistic regression (LR()), `Path2Str` and bag

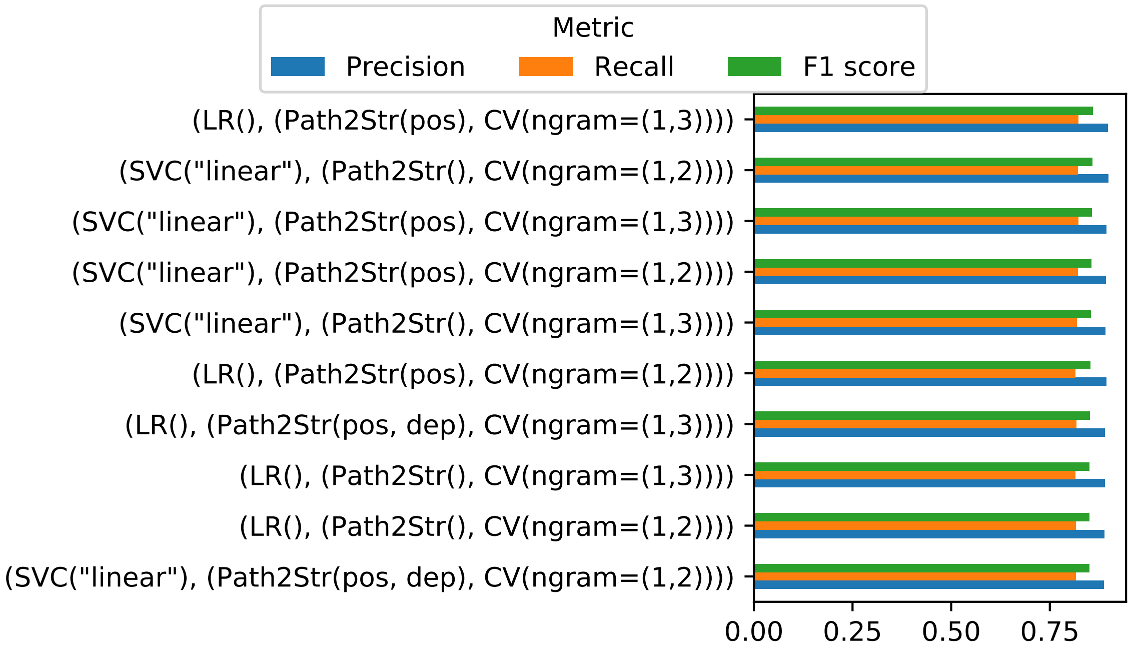


Figure 15: Classifier ranking according to F1 score. The best results for F1 score (88%), precision (92%) and recall (84%) are achieved with feature-classifier-combinations of `Path2Str`, bag of words with 2- or 3-grams (CV(ngram=(1,2)) or CV(ngram=(1,3))) and logistic regression (LR()) or SVM with a linear kernel (SVC("linear")). PoS-tags and dependency labels have little influence on the scores. The performance of all feature-classifier-combinations is shown in figure 16.

of words with 2- or 3-grams (CV(ngram=(1,2)) or CV(ngram=(1,3))) (figure 15). It seems better to use 2- or 3-grams instead of the default unigrams (cf. section 3.3). However, it does not make a noticeable difference whether 2- or 3-grams are employed (figure 15). Comparing the F1 score to precision and recall also shows the little influence PoS-tags and dependency labels have on the scores (figure 15).

The RE performance is limited by the NER performance. The NER pipeline performs with an accuracy of 88%, a recall of 97% and a precision of 98% (figure 12). The best feature-classifier-combination of the RE pipeline achieves a recall of 84% and a precision of 93% which leads to an F1 score of 88% (figure 15).

The classifiers LR() (figure 16, top) and SVC("linear") (figure 16, middle) show similar behavior in stark contrast to the classifier SVM with radial basis function kernel (SVC("rbf")) (figure 16, bottom).

First, the classifier plots of SVC("linear") and LR() are discussed. Precision and recall of both SVC("linear") and LR() are clustered into four groups (figure 16, top and middle). The worst feature is `Doc2Vec`. As `Doc2Vec` is known to only achieve good scores for a

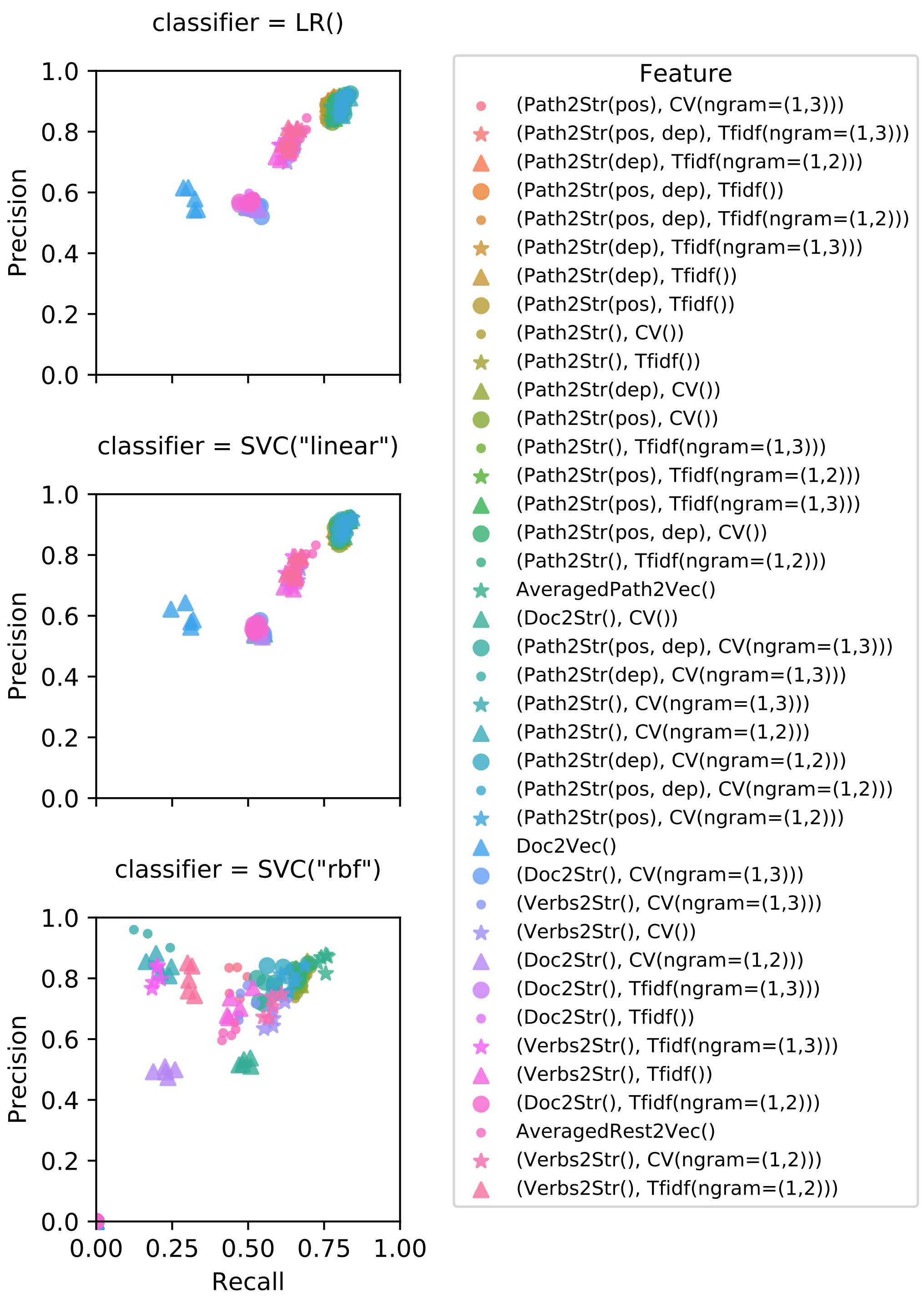


Figure 16: Overview of the performance of all feature-classifier-combinations. The best combinations employ `Path2Str`, bag of words with 2- or 3-grams (CV(ngram=(1,2)) or CV(ngram=(1,3))) and logistic regression (LR()) or SVM with a linear kernel (SVC("linear")). The top ten performers are ranked in figure 15.

large data set, probably this feature is not adequate for the CNT data set (cf. section 3.3). Doc2Str achieving an F1 score of approximately 54% belongs to the next best group. Providing the whole design without making use of the knowledge from the NER pipeline results to be a weak feature for the given task (cf. section 4.2). Verb2Str (F1 score of up to 71%) belongs to the same group as AveragedRest2Vec. The performances of the classifiers is improved by combining the string transformers with a vectorizer using 2- or 3-grams. The best results are achieved with Path2Str combined with any vectorizer on 2- or 3-grams. Almost as good performs AveragedPath2Vec achieving an F1 score up to 82% (figure 16, top and middle).

The success of n-grams can be explained in the following way: The majority of all relations based on verbs in English can be described with the pattern $(NE_1, verb, NE_2)$. Therefore it becomes likelier for the classifier to find these relations if n-grams are provided as features (cf. section 3.7).

The results achieved by the SVC("rbf") classifier are not as good as those of the other two classifiers: The best feature for SVC("rbf") is AveragedPath2Vec with an F1 score of about 80% (figure 16, bottom). The features are not clustered into groups. Doc2Vec, Doc2Str with any variation of TF-IDF and even some Path2Str variations score 0%. All things considered, SVC("rbf") does not provide any expressive results for RE on the CNT designs.

All discussed scores have been achieved after looping over the RE workflow twice (cf. section 4.2). Given the output of the cross-validated grid search, RE was performed in each loop with the best feature-classifier-combination. By analyzing the False Positives, the annotated relations were manually improved in each loop which had also an impact on the classifier's performance (cf. section 4.2).

Due to the lack of annotated training data, the RE pipeline was not run on external data sets. However, it is possible to predict relations for external designs given the RE classifier has been trained on the CNT designs annotated with relations. Therefore it can be said that the RE implementation generalizes to other data sets.

7 Summary and Outlook

The goal of this bachelor thesis was to take first steps in enhancing search on numismatic descriptions. This definition of task has been adapted by employing *Natural Language Processing (NLP)*. The discussed approaches of *named entity recognition (NER)* and *relation extraction (RE)* are based on supervised methods (cf. section 3.7). The developed implementations automatically learn how to recognize named entities and extract relations based on the *Corpus Nummorum Thracorum (CNT)* data set. Both implementations are able to adapt to other data sets without any further adaptations. For the NER pipeline, generalizability has been shown on the external data sets of *Online Coins of the Roman Empire (OCRE)* and *Coinage of the Roman Republic Online (CRRO)* (cf. section 6.1). Given the RE classifier is trained on the CNT designs, it is able to predict relations for external data sets like OCRE or CRRO. Thus the RE pipeline also generalizes to other data sets (cf. section 6.2).

The most sophisticated search approach so far is the *Digital Iconographic Atlas of Numismatics in Antiquity (DIANA)*. Being knowledge-based, it demands a high amount of manual labor and does not generalize towards other data sets (cf. section 2.2). In comparison, the approach developed in this thesis learns automatically because it is based on supervised methods. Second, it is flexible because new entity types can be added and all combinations of relations between (PERSON, OBJECT) pairs can be extracted. Thirdly, it generalizes to other data sets because it is automated.

Given the small data set of CNT designs, understanding RE as a multiclass classification problem is a new approach (cf. section 3.7). This approach developed in this thesis is based on a supervised method, but by annotating relations only, it requires less manual labor than supervised methods usually do (cf. section 4.2). However, the number of relations is fixed, so no new relations can be extracted by the classifier. After having analyzed different feature-classifier-combinations, it is recommended to employ `Path2Str`, bag of words with 2- and 3-grams as features and logistic regression or SVM with a linear kernel as classifiers (cf. section 6.2).

The classifiers' performance in the NER and in the RE pipeline have been evaluated by using the metrics accuracy, precision, recall and F1 score (cf. section 3.4). In the NER pipeline, an accuracy of 88%, a recall of 97% and a precision of 98% has been achieved (cf. section 6.1). In the RE pipeline, a recall of 84%, a precision of 93% and an F1 score of 88% has been achieved (cf. section 6.2). Thus both classifiers perform very well. Nevertheless, the metrics leave room for further optimization.

In future studies, the following enhancements and extensions could be of interest:

Performance Optimization To improve the implementation, the NE recognizer's and the RE classifier's performances could be optimized.

Search Hierarchy To build a search hierarchy, the labels PERSON, OBJECT, ANIMAL and PLANT in the NER implementation could be subdivided. For example, the label PERSON could be distinguished in "historical person" and "deity".

Search Hierarchy and Further Relations To build a search hierarchy and detect further relations in the CNT designs, other relations than those already identified could be examined, e.g. the relations of persons and body parts. For

> "Apollo seated left on omphalos, holding bow in right hand.",

it might be interesting to ask in which hand Apollo holds the bow to distinguish "right" and "left hand".

Further Named Entity Combinations To examine further combinations of named entities in the CNT designs, the already identified relations could be employed for pairs such as (PERSON, ANIMAL), (PERSON, PLANT) or (PERSON, PERSON). Possibly, further relations would be detected thereby.

Generalizability To probe, how well the RE implementation generalizes towards other data sets, external data sets like OCRE or CRRO could be annotated manually with the corresponding relations. Thus the RE classifier could be trained and evaluated on an external data set.

Acknowledgments

I would like to thank Prof. Dott.-Ing. Roberto V. Zicari for supervising this bachelor thesis and Prof. Dr. Christian Chiarcos for co-supervision and sharing his expertise on Natural Language Processing. I especially thank my direct instructor Dr. Karsten Tolle for conceptualizing this thesis, for all the helpful discussions and for the always friendly working atmosphere. I would also like to thank Sebastian Gampe for his archaeological expertise and his assistance in annotating training data for the NER pipeline.

Appendix

A data medium containing the implementations discussed in this bachelor thesis is annexed. The corresponding code can be found in the cnt package. The usage of this code, especially the setup of the NER and RE pipeline, is documented in the attached Jupyter notebooks. Furthermore, the database dump and the file for relation annotation are attached.

References

Bird, Steven, Ewan Klein, and Edward Loper (2009). *Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit.* URL: http://www.nltk.org/book/.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning.* Springer.

Celesti, Antonio et al. (2017). "An Innovative Cloud-Based System for the Diachronic Analysis in Numismatics". In: *Journal on Computing and Cultural Heritage* 10.4, 23:1–23:18. ISSN: 1556-4673. DOI: 10.1145/3084546. URL: http://doi.acm.org/10.1145/3084546.

Honnibal, Matthew, Yoav Goldberg, and Mark Johnson (2013). "A Non-Monotonic Arc-Eager Transition System for Dependency Parsing". In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pp. 163–172.

Konstantinova, Natalia (2014). "Review of Relation Extraction Methods: What Is New Out There?" In: *Analysis of Images, Social Networks and Texts - Third International Conference, AIST 2014, Yekaterinburg, Russia, April 10-12, 2014, Revised Selected Papers*, pp. 15–28. DOI: 10.1007/978-3-319-12580-0_2. URL: https://doi.org/10.1007/978-3-319-12580-0_2.

Lampple, Guillaume et al. (2016). “Neural Architectures for Named Entity Recognition”. In: *CoRR* abs/1603.01360. arXiv: 1603.01360. URL: http://arxiv.org/abs/1603.01360.

Le, Quoc V. and Tomas Mikolov (2014). “Distributed Representations of Sentences and Documents”. In: *CoRR* abs/1405.4053. arXiv: 1405.4053. URL: http://arxiv.org/abs/1405.4053.

Mikolov, Tomas et al. (2013). “Distributed Representations of Words and Phrases and Their Compositionality”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’13. Lake Tahoe, Nevada: Curran Associates Inc., pp. 3111–3119. URL: http://dl.acm.org/citation.cfm?id=2999792.2999959.

Muller, Andreas C. and Sarah Guido (2017). *Introduction to Machine Learning with Python. A Guide for Data Scientists*. O’Reilly. Chap. 7.

Pedregosa, F. et al. (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Sarkar, Dipanjan (2016). *Text Analytics with Python: A Practical Real-World Approach to Gaining Actionable Insights from your Data*. Apress. Chap. 4.