Ruprecht-Karls-Universität
Heidelberg

Neuphilologische Fakultät
Seminar für Computerlinguistik

Magisterarbeit

Classification of named entities in a large multilingual resource using the Wikipedia category system

Autor:
Johannes Knopp

Gutachter:
Prof. Dr. Anette Frank
Dr. Stefan Riezler

Heidelberg, den 25. Januar 2010

Danksagung

Meinen Eltern gilt der größte Dank. Sie haben mich immer auf jegliche Weise unterstützt und mir das Studium ermöglicht. Danke.

Für die zahlreichen fruchtbaren Diskussionen, die Zusammenarbeit in technischen Dingen und die einhergehenden gemeinsamen Pausen danke ich Wolodja Wentland. Andere Menschen, denen ich für ihre hilfreichen Hinweise bei der Arbeit an der Magisterarbeit danken möchte, sind Sascha Fendrich, Britta Zeller, Matthias Hartung und mein Bruder Sebastian Knopp. Ein weiterer Dank gilt Nora Borrusch, die dafür Sorge getragen hat, dass der entstandene Text den Regeln der englischen Grammatik entsprechen sollte. Des Weiteren bedanke ich mich bei Prof. Dr. Anette Frank für die Betreuung dieser Arbeit und für die langjährige Unterstützung im Studium.

Table of Contents

1 Introduction

Over the last 15 years the role of named entities became more and more important in natural language processing (NLP). Their information is crucial for tasks in information extraction like coreference resolution or relationship extraction. As recent systems mostly rely on machine learning techniques, their performance is based on the size and quality of given training data. This data is expensive and cumbersome to create because usually experts annotate corpora manually to achieve high quality data. As a result, these data sets often lack coverage, are not up to date and are not available in many languages.

To overcome this problem, semi-automatic methods for resource construction from other available sources were deployed. One of these sources is Wikipedia, a free collaboratively created online encyclopedia, which was explored for several NLP tasks over the last years. Although it is not created by linguists, meta information about articles such as translations, disambiguations or categorisations are available. In addition, Wikipedia is growing fast: it is available in more than 260 languages and contains more than three million articles in the English version.

The structural features, its size and multilingual availability provide a suitable base to derive specialised resources that can be used as training data for machine learning. One of them is *HeiNER – the Heidelberg Named Entity Resource* (Wentland et al., 2008). HeiNER contains a huge multilingual collection of named entities including their contexts taken from Wikipedia. However, there is one disadvantage: it has no knowledge of which type its named entities are. Hence, the idea of this thesis is to add the named entity types *Person*, *Organisation*, *Location* and *Miscellaneous* to HeiNER's entries.

Wikipedia's Category system is utilised to solve this problem. We identify categories that unambiguously match a named entity type in order to classify all articles found in them automatically. Counting the categories of these new classified articles results in named entity type vectors that are used to classify the yet unlabelled named entities that are members of HeiNER.

The structure of this thesis is as follows: Section 2 provides the definition of named entities, which tasks and conferences dealt with them and which types are used. Section 3 continues to tell about machine learning techniques that have been applied to NE classification in prior work. Wikipedia as a resource and referential work is presented in 4. Additionally, the Wikipedia API *mwdb*, which was written in the course of this thesis, will be shown. How the classification of HeiNER's named entities was carried out is explained in section 5 followed by its evaluation in 5.3. The resulting conclusions and an outlook are given in section 6. Further information can be found in the appendix.

2 Named Entities

The term *named entity* (NE) was first introduced to the Natural Language Processing community at MUC-6, i.e. the Sixth Message Understanding Conference (Grishman and Sundheim, 1996). It refers to expressions describing entities with a rigid designator[1] like persons, locations and organisations called *ENAMEX* ("entity name expression"). MUC-6 also included the numeric expressions (*NUMEX*) money and percent and introduced time expressions (*TIMEX*) for time and date. An example of annotated MUC-6 data can be seen in figure 1; later work introduced other named entity types (see chapter 2.2).

The task of identifying named entities within texts is called named entity recognition (NER) and labelling them with their type is known as named entity classification (NEC), which combines results in named entity recognition and classification (NERC)[2]. NERC is used in different fields like information retrieval or question answering.

This chapter presents a brief overview of the development of named entity recognition over the last 14 years. After explaining in more detail what a named entity is, we document the shift from pattern based systems to automatic machine learning and eventually list some of the more fine-grained named entity types that were proposed.

[1]The term rigid designator was introduced by the philosopher Saul Kripke in his work Naming and Necessity (Kripke, 1972): a term referring to the same thing in all possible worlds in which that thing exists is a rigid designator. The presented work will not deal with the philosophical view of named entities and MUC-6 did not use the term itself, but it gives the right idea of what a named entity is.

[2]Often NER is used interchangeably with NERC because named entity recognition without classification does not make sense in many cases.

Mr. <ENAMEX TYPE=”PERSON”>Dooner</ENAMEX> met with <ENAMEX TYPE=”PERSON”>Martin Puris</ENAMEX>, president and chief executive officer of <ENAMEX TYPE=”ORGANIZATION”>Ammirati & Puris</ENAMEX>, about <ENAMEX TYPE=”ORGANIZATION”>McCann</ENAMEX>’s acquiring the agency with billings of <NUMEX TYPE=”MONEY”>$400 million</NUMEX>, but nothing has materialized.

Figure 1: Sample of MUC-6 annotation

NERC has developed into a task of its own in Natural Language Processing because named entities contain a great deal of information. In order to understand a text or represent its content correctly, systems must be able to answer questions like who, what, when and where. Most of the answers are named entities: persons or organisations act in locations at a particular time. These expressions do not appear in ordinary dictionaries, which makes automatic detection and classification necessary. Consider this sentence found in the Washington post:[3]

> Pope Benedict XVI waded into a crowd of well-wishers in Rome on Sunday, just days after he was knocked down by a woman at a Christmas Eve Mass. It was the 82-year-old pontiff’s first appearance outside the Vatican since the attack [...]

The news’ value lies in the named entities: “Pope Benedict XVI” is the acting person, “Rome” his location and “Sunday” the date of his action. The action itself – “wading into a crowd of well-wishers” – is not a named entity. The last named entity in this example is “Vatican“.

Note that “the 82-year-old pontiff” in the second sentence is a reference to the named entity “Pope Benedict XVI”. This implies that the expressions refer to the identical entity, and thus one could think that “the 82-year-old pontiff” should be mentioned as a named entity as well. But that fact is only inferred from the context, as “the 82-year-old pontiff” could mean any bishop of that age and is only used as a reference to a named entity. That is why the expression could

[3]Found on December 27th at `http://www.washingtonpost.com/wp-dyn/content/article/2009/12/27/AR2009122702024.html`.

not be considered as a named entity when standing alone. It does not have a rigid designator and thus is not a named entity, even in this context. Identifying referring expressions in texts is a task called coreference resolution, but this work does not deal with it as it is a research field of its own.

Not covered by the MUC-6 named entity types is “Christmas” although it has a rigid designator: the birthday of Jesus. We will see in section 2.2 that other definitions include this case, but also that it can be hard to draw a line between named entities and non-named entities. For example ask yourself if you would consider “December“ to be a named entity. It has a rigid designator – “the twelfth month of the Gregorian calendar” –, but it is not clear without context which year is meant, which adds some incertitude; so we have to ask what “rigid“ really means. Nadeau and Sekine (2007) stated on that topic:

> Rigid designators include proper names as well as certain natural kind terms like biological species and substances. There is a general agreement in the NERC community about the inclusion of temporal expressions and some numerical expressions such as amounts of money and other types of units. While some instances of these types are good examples of rigid designators (e.g., the year 2001 is the 2001st year of the Gregorian calendar) there are also many invalid ones (e.g., in June refers to the month of an undefined year – past June, this June, June 2020, etc.). It is arguable that the named entity definition is loosened in such cases for practical reasons. (Nadeau and Sekine, 2007, page 3)

We see that NER is not a trivial task, which is underlined by Fleischman and Hovy (2002), who point out that, for instance, classifying entities into subtypes by their context can be hard even for humans.

2.1 History of named entity tasks

Tasks dealing with named entities developed over time, starting with MUC-6 in 1996. The interest at that time focused on information extraction (IE) tasks to find information on companies or defence-related activities in newspaper articles.

A multilingual setup was first introduced by the Multilingual Entity Task (MET) (Merchant et al., 1996) with three languages: Spanish, Japanese and Chinese. In 1999, MUC-7 and MET-2 (Chinchor, 1999) added another multilingual setup using Japanese and Chinese as well. Up to this point the training and test sets were restricted to one domain. This changed with IREX (Sekine and Isahara, 2000), which had a restricted and an unrestricted domain task. Sekine and Isahara report three kinds of systems: 1. pattern based systems, in which patterns are written by humans; 2. also pattern based, but the patterns are extracted from a tagged corpus by some automatic means; and 3. fully automatic systems, which do not use explicit patterns. None of the three types was outperforming the others, but with the CoNLL shared tasks presented by Sang (2002) and Sang and Meulder (2003), automatic systems came to the fore on the task of language-independent named entity recognition. Since 2000, the Language Resources and Evaluation Conference (LREC)[4] also provides a platform for named entity tasks and resources.

A new task definition appeared in the Automatic Content Extraction (ACE) Program including "recognition of entities, not just names" (Doddington et al., 2004, page 1). In other words: NER and tracking combine named entity recognition with a coreference task. As more tasks on named entities were carried out, different evaluation schemata grew. They rely on boundary detection (Is a [multiword] named entity found correctly?) and classification (Is the correct named entity type assigned?) to compute precision, recall and f-score of results on a test set. The schemata differ in dealing with incorrect type assignments and partial matches. An overview is given in Nadeau and Sekine (2007, chap. 4).[5]

All in all the complexity of NE tasks have grown over the years and the applied systems shifted from pattern based approaches to machine learning techniques that require training data. The underlying idea is the construction of learners that are not suited for just one language, but try to find generalisations that help to analyse a set of different languages. This development motivated researchers to focus on the creation of useful resources and comparable evaluations that are discussed on conferences like LREC.

Likewise to the development of named entity tasks, more named entity types

[4]`http://www.lrec-conf.org`

[5]Another read on this topic is a blog post by Christopher Manning in August 2006: "Doing named entity recognition? Don't optimize for F_1" (`http://nlpers.blogspot.com/2006/08/doing-named-entity-recognition-dont.html` , last checked on January 2_{nd} 2010).

were introduced. The following section provides an insight into some of the explored types.

2.2 Types of named entities

Until now we have mentioned that named entities are distinguished by their type and that MUC-6 introduced the ENAMEX, TIMEX and NUMEX types covering proper names, time and numerical expressions. Other works divided these types or classes[6] in more fine-grained subtypes. *Location* can be split into city, state, country etc. (Fleischman, 2001) and a *Person* can be e.g. an entertainer, a politician or a scientist amongst others (Fleischman and Hovy, 2002; Lee and Lee, 2005). Even mixtures were introduced in the Automatic Content Extraction (ACE) Program (Doddington et al., 2004), where facility subsumes entities that are of both types *Location* and *Organisation*, and a Geo-Political entity represents a location having a government such as a city or country. The CoNLL conferences (Sang and Meulder, 2003) filed proper names which do not fit into the ENAMEX type schema under *Miscellaneous*, including events, languages and types (not brands) of objects. As this work will adhere to the CoNLL definition later (cf. chapter 5), table 1 shows the complete annotation guidelines.[7]

Less general types were sometimes used for certain domains, e.g. e-mail address and phone number (Witten et al., 1999) or in the field of bioinformatics protein, DNA, RNA, cell line and cell type (Shen et al., 2003). Aside from the given types exists the open domain NERC (Alfonseca and Manandhar, 2002; Evans, 2003), which does not limit the possible types to extract.

The various mentioned types show that the decision what can be regarded as a named entity is not answered in the same way in the different setups. Nevertheless, automatic learning systems have to deal with this range of possible named entity types. Automatic learning techniques for NER are presented in the following chapter.

[6]The terms type and class are used interchangeably in this work.

[7]The table was created according to `http://www.cnts.ua.ac.be/conll2003/ner/annotation.txt` (last checked on January 24_{th} 2010).

Named Entity Type	**Generic Term**	**Examples**
Person	first, middle and last names of people, animals and fictional characters	
	aliases	
Organisation	companies	press agencies, studios, banks, stock markets, manufacturers, cooperatives
	subdivisions of companies	newsrooms
	brands	–
	political movements	political parties, terrorist organisations
	government bodies	ministries, councils, courts, political unions of countries (e.g. the *U.N.*)
	publications	magazines, newspapers, journals
	musical companies	bands, choirs, opera companies, orchestras
	other collections of people	sports clubs, sports teams, associations, theaters companies, religious orders, youth organisations
Location	roads	streets, motorways
	trajectories	–
	regions	villages, towns, cities, provinces, countries, continents, dioceses, parishes
	structures	bridges, ports, dams
	natural locations	mountains, mountain ranges, woods, rivers, wells, fields, valleys, gardens, nature reserves, allotments, beaches, national parks
	public places	squares, opera houses, museums, schools, markets, airports, stations, swimming pools, hospitals, sports facilities, youth centers, parks, town halls, theaters, cinemas, galleries, camping grounds, NASA launch pads, club houses, universities, libraries, churches, medical centers, parking lots, playgrounds, cemeteries
	commercial places	chemists, pubs, restaurants, depots, hostels, hotels, industrial parks, nightclubs, music venues
	assorted buildings	houses, monasteries, creches, mills, army barracks, castles, retirement homes, towers, halls, rooms, vicarages, courtyards
	abstract "places"	(e.g. *the free world*)
Miscellaneous	words of which one part is a location, organisation, miscellaneous, or person	–
	adjectives and other words derived from a word which is location, organisation, miscellaneous, or person	
	religions	–
	political ideologies	–
	nationalities	–
	languages	–
	programs	–
	events	conferences, festivals, sports competitions, forums, parties, concerts
	wars	–
	sports related names	league tables, leagues, cups
	titles	books, songs, films, stories, albums, musicals, TV programs
	slogans	–
	eras in time	–
	types (not brands) of objects	car types, planes, motorbikes

Table 1: The CoNLL 2003 annotation guidelines.

3 Learning and Classification

As we have seen, the task of NERC has developed over the years and likewise have the applied methods. This chapter will present a general overview of learning methods that were used to classify named entities. The intention is to describe studied approaches to the task as opposed to the strategy that is applied in the presented work because they provide the motivation for it. One major goal of the classification of HeiNER's named entities is to make training data available to machine learning systems.

When starting with NERC, researchers had to think of ways to find unknown words in texts. Named entities are unknown words because they can not be looked up in any ordinary lexicon. To identify them in a machine learning scenario, a set of distinct features is needed to tell positive and negative examples apart. First, we introduce features for automatic NERC in the following subsection. After that we concentrate on learning algorithms that rely on them.

3.1 Features

Features can be described as properties of texts. They provide information about words, sentences or even complete documents. Compositions of features are called feature vectors, which are an abstract representation of the underlying text. This abstraction allows to find generalisations which is crucial for automatic classification systems.

For example, consider these four common word level features: (I) the lowercase word, (II) word length in characters, (III) capitalisation of the first character, and (IV) position of the word in the sentence. The sentence "Heidelberg is a city in Baden-Württemberg, Germany.", taken from the Wikipedia article about Heidelberg[8] would be represented as a set of feature vectors in this way (leaving out the punctuation):

[8]http://en.wikipedia.org/w/index.php?title=Heidelberg&oldid=338390932

```
<``Heidelberg'', 10, true, 1>, <``is'', 2, false, 2>,
<``a'', 1, false, 3>, <``city'', 4, false, 4>,
<``in'', 2, false, 5>, <``Baden-Württemberg'', 17, true, 6>,
<``Germany'', 7, true, 7>
```

The challenge for a NERC system is to find named entity candidates and assign their respective types to them with the help of the features in the vectors. In order to do this they have to know rules which they can apply. In this example the candidate selection rule could be "if the word is capitalised, take it as a candidate", and the – rather simple – classification rule could be "if the word has more than 6 characters, it is a location". For real-world sentences these rules can get very complex and usually are not as explicit as the examples given above. More on the learning of these rules is explained in subsection 3.2. Nadeau and Sekine (2007) summarise four kinds of features that are presented in the subsequent sections:[9]

1. Word based features
2. List-lookup features
3. Document features
4. Corpus based features

3.1.1 Word-level Features

Considering a single word, there are some character based features we can look at:

- Case – Capitalisation of first/some/all letters (e.g. "Heidelberg", "YouTube", "IBM")
- Punctuation – period at end (e.g. "Mr.") or in the word (e.g. "I.B.M."), internal apostrophe, hyphen or ampersand (e.g. "O'Reilly")
- Significant character – possessive mark, first person pronoun, Greek letters (e.g. "'s", "α", "β", ...)

[9]The features are presented as in the original paper, but the order has been rearranged sometimes. Most of the examples are borrowed, too, but will not be labelled explicitly to keep up the readability.

- Digit – cardinal and ordinal, Roman number, word with digits (e.g. “W3C”, “26C3”...)

All of them are indicators for different kinds of information: Punctuation is a sign for abbreviations, capitalised characters on other positions than the beginning of the sentence indicate a proper name, and digits can express many different types of information like a year (two and four digits) or a fraction (digits in combination with a percentage sign). Other features need some analysis of the word first:

- Morphology – prefix, suffix, singular version, stem, common ending
- Part-of-Speech – proper name, verb, noun, foreign word
- Function – alpha, non-alpha, n-gram, lowercase/uppercase version, pattern, summarised pattern, token/phrase length

Some of these features need explanation. Affixes and stems are computed by a morphological analyser, and some of the prefixes can help to identify common word endings. For example “-ish” and “-an” hint at nationalities and languages (Spanish, Danish, Romanian), human professions often end in “-ist” (journalist, dentist), and organisation’s names often end in “-ex”, “-tech” or “-soft” (Bick, 2004).

The function-features process a given word to represent it in a new way. For example, the non-alpha representation of “A.T.&T.” is “..&.”. A so-called pattern is a mapping of characters to a smaller set of character types to receive a uniform word representation. For instance, uppercase characters could be mapped to “A”, lowercase to “a”, punctuation to “-” and digits to “0”. The mapping of “G.M.” would be “A-A-”, the one of “Machine-223” is “Aaaaaaa-000”. The summarised pattern compresses repeated characters, which still results in “A-A-” for “G.M.”, but “Machine-223” yields “Aa-0” as a summarised pattern.

3.1.2 Document and Corpora Features

Document features are not just properties of the content of a whole document, but also rely on its structure and composition. Corpora are large collections of documents, and statistics computed on them are used as features in the NERC task, too:

- Multiple occurrences – other entities in the context, uppercased and lowercased occurrences, anaphora and coreference
- Local syntax – enumeration, apposition
- Meta information – URI, e-mail header, XML section,bulleted/numbered lists, tables, figures
- Corpus frequency – word and phrase frequency, co-occurrences, multiword unit permanency

An entity often occurs in a document several times, especially if it is a named entity. A problem is that it might be referred to in different ways, for example a person might be mentioned by their name, their title or an anaphora (most commonly a pronoun). "Coreference [...] describes the relation that holds between two expressions that refer to the same entity"(Denis, 2007, page 3). If it is possible to find all expressions referring to the same entity, we can extract information from their contexts and derive deduce features from them.

Well structured meta information of a document can be used directly; for example it is possible to find people's names in email headers or extract the location from the beginning of a news article.

Statistics about corpus frequencies of words, phrases and co-occurrences help to estimate the importance of words found in a context.

3.1.3 List-Lookup Features

External knowledge sources are often used to look up candidates and use the retrieved information to improve the classification performance. All sources are subsumed under the term "list" here, although they might include more sophisticated information such as taxonomic structure. This holds for ontologies such as WordNet (Fellbaum, 1998), in particular.

- General list – general dictionary, stop/function words, capitalised nouns (e.g. "January", "Monday")

- List of entities – organisations, governments, airlines, first names, last names, celebrities
- List of entity cues – typical words in organisations, person titles, name prefixes, post-nominal letters, location-typical words, cardinal points

A word at the beginning of a sentence can be looked up in a general dictionary to decide whether it is commonly written capitalised or if it should be considered a named entity candidate. A list of entities simply allows looking up named entities. To recognise organisations, a list of words that typically appear in organisation names are useful. For example, these names often include "-inc." or "-corp" (Rau, 1991). WordNet is used in Fleischman and Hovy (2002) to add weights to topic signature features.

So now that a general overview about features is given, the next chapter will deal with the automatic learning systems that take advantage of them.

3.2 Learners

Over the years, beginning from MUC-6 to recent tasks on NERC, new methods from the machine learning field became more and more popular, leaving behind systems which use handcrafted rules. Machine learning techniques allow the automatic induction of rule-based systems or sequence labelling algorithms from allocated training data. This is achieved by analysing the discriminative features of positive and negative examples. Similar cases and repetitions occurring in the data are merged into rules and hence gain abstraction over concrete examples. Three different types of learning methods can be distinguished by their requirements for the training data:

1. Supervised learning
2. Semi-supervised learning
3. Unsupervised learning

The following sections will present these methods in the field of NERC.

3.2.1 Supervised learning

Supervised learning methods rely on large annotated training corpora. They try to identify the rules behind the feature vectors to be able to classify unseen data afterwards. Several algorithms have shown to be competitive. They are summarised and provided with example references mostly taken from the CoNLL shared task 2003 in this section:[10]

A *Hidden Markov Model* (HMM) consists of a set of (hidden) states, transition probabilities between them and observed output depending on the current state. In the named entity classification setup this means that words in a text are the observed output. The most probable sequence of hidden states (the named entity types) that produced these words has to be inferred with help of the knowledge of the annotated data. HMMs were first used for NERC by Bikel et al. (1997) who presented a model where the internal states of the HMM matched the named entity types of MUC-6 and had three additional states: "start-of sentence" and "end-of-sentence" plus the state "not-a-name" for common words. They applied it on English MUC-6 and Spanish MET data and reported f-scores of 90 and above for this probabilistic model.

Sang and Meulder (2003) reported that most of the systems used in the CoNLL shared task implemented *Maximum Entropy Models* (Berger et al., 1996). For example, Bender et al. (2003) computed the probabilities of a named entity tag for a given word by factorising "the posterior probability and determine the corresponding NE tag for each word of an input sequence"(Bender et al., 2003, page 1). They assumed that the decisions only depend on a limited window of two predecessor tags.

Another automatic learning algorithm is a *decision tree*:

> A decision tree can classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. Each node in the tree specifies a test of some attribute [i.e. a feature] of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute. (Mitchell, 1997, chap. 3.1)

[10]This selection does not claim to be complete.

For the example from page 9, the root of the decision tree of the rules would ask for the value “capitalised”, leading to the classification “not a named entity” when descending branch “no” and otherwise going for another node “len(word) > 6” whose two branches lead to “organisation” or “not a named entity” for the answers “yes” and “no”. As you can see a nice property of decision trees is that they can be represented as sets of if-then rules to improve human readability.

Decision trees were used for NERC by Szarvas et al. (2006) in a multilingual setup. The decision tree learning algorithm C4.5 (Quinlan, 1993) is used in combination with AdaBoost (Carreras et al., 2002), an algorithm that generates a set of classifiers (of the same type) by applying bootstrapping[11] on the original training data set and deciding based on their votes. They report 2.32% error reduction on the CoNLL shared task relative to the best model tested on the same data. This improvement is significant because the best results already were in the region of 88% f-score (English test set).

Support Vector Machines (SVM) are a classification method that tries to find a *hyperplane* to separate positive from negative training examples (the feature vectors) of one type in a vector space. This allows the classification of new examples by judging on which side of the hyperplane they reside and how far they are away from the margin. Mayfield et al. (2003) participated in the CoNLL 2003 NERC task with a SVM based system and reached the 7th (4th) position for the English (German) language with an f-score of 84.67% (69.96%).

Another CoNLL participant were McCallum and Li (2003) who used *Conditional Random Fields* (CRF) (Lafferty et al., 2001): “Conditional Random Fields [. . .] are undirected graphical models used to calculate the conditional probability of values on designated output nodes given values assigned to other designated input nodes”(McCallum and Li, 2003, p. 1). In contrast to HMMs they are aware of the complete sequence of input nodes. McCallum and Li (2003) try to maximise the performance of their system by feature induction. That means features are only added to the system if they seem to be beneficial because a “large number of features can be prohibitively expensive in memory and computation”(McCallum and Li, 2003, p. 3). In order to do that they consider a set of proposed new features, add the candidates that will maximise the log-likelihood of the correct

[11]Cf. semi-supervised learning on page 15.

state paths and then train their weights. They report an overall f-score of 84.04% on the English data set using 6,423 features. The same system using a fixed set of conjunction patterns without feature induction results in a ∼10% lower f-score of 73.34% using about 1 million features. The lesson learned is that the selection of features can change the performance of a system considerably. The quality of the feature vector is more important than its size.

A popular system using CRFs is the Stanford Named Entity Recogniser[12] (Finkel et al., 2005).

Note that the best participant of the CoNLL 2003 shared task – Florian et al. (2003) – used a combination of classifiers: a robust linear classifier, maximum entropy, transformation-based learning and a Hidden Markov Model. Additionally, the system used a gazetteer[13] and reached the first position for both English and German with an f-score of 88.76% and 72.41% respectively. Consequently, it seems to be a good idea to take advantage of the different properties of supervised learners to build a superior system out of a combination of methods.

In conclusion, there are many ways to train a supervised learner and they all have in common that their performance correlates with the amount of available training data or as Nadeau and Sekine (2007) put it: “The main shortcoming of supervised learning is the requirement of a large annotated corpus”(Nadeau and Sekine, 2007, page 4).

3.2.2 Semi-supervised learning

Because of the disadvantage of supervised learners needing large annotated corpora, semi- or weakly supervised learners were introduced. I. e. a system starts with a small data set and tries to enhance it with new classified instances from unlabelled data and then starts the process again. This iterative approach is called bootstrapping and surmounts the need for large corpora which are expensive and tedious to create.

Bootstrapping is used by Kozareva (2006) on the CoNLL data for two tasks: (i)

[12]`http://nlp.stanford.edu/software/CRF-NER.shtml`

[13]specialised lists describing an isa-relation. E.g. “Heidelberg” would be found in a location-gazetteer.

to automatically induce a gazetteer of person names[14] (ii) to compare a supervised and a semi-supervised approach for NERC.

To create the gazetteer of person names, he takes a large unannotated corpus and tries to find and validate common name patterns. Person names are found by starting with the frequent Spanish first name "José" and looking for combinations with last names in the corpus. Unseen name combinations with a frequency of more than ten are split in first and last name and then added to the person-gazetteer together with this information. The next iterations look for new name combinations of first and last names with the required frequency and do the same until no more new instances are found. In this way many names can be found starting from only one seed entry. A similar approach is used in this work to gather Wikipedia categories (cf. chapter 5.1.1, page 29 ff.).

NERC is accomplished in a supervised and a semi-supervised setup. Both use two learners: instance based[15] and decision trees. In the supervised version all of the training data is labelled with correct named entity types and the learners are trained on it. The semi-supervised setup is only provided with a set of annotated seed samples to start bootstrapping. The training is done on the same data as in the supervised setup but this time with hidden labels. For that reason the learners classify the unlabelled training data, take the most confident classifications into account and then start all over again until a maximum of 25 iterations is reached.

Both setups were executed with and without including the induced gazetteers as a list-lookup feature. The resulting systems were evaluated on an annotated test set. The supervised method beats the bootstrapping approach with about 20% higher f-score. Still, the author states that the results for semi-supervised learning are promising when thinking of tasks where no annotated data is available. The automatically induced gazetteers prove to be useful increasing the quality of classifications about 4%. If the named entities in HeiNER were classified they would provide comparable gazetteers and additional training data.

[14]He also creates a gazetteer for locations but with a technique related to the idea of pattern based information extraction first introduced by Hearst (1992) instead of bootstrapping.

[15]A method where all training instances are stored and classification is done by computing its relationship to to the previously stored examples (Mitchell, 1997, chap. 8).

3.2.3 Unsupervised learning

In *unsupervised learning* the training data is not labelled and the classes that can be learned are unknown to the system (Manning and Schütze, 2002, p. 232). Thus, unsupervised learners try to find similar instances and cluster them by their features into undefined classes.[16] This does not mean that the output of such a classifier cannot be interpreted. For example Shinyama and Sekine (2004) uses the simultaneous appearance of a phrase in several news articles to identify rare named entities in an unsupervised manner.

Sometimes the definition of unsupervised learning is incorrectly reduced on stating that the data a system uses is not labelled. For example Collins and Singer (1999) describe their system to be unsupervised although they start with a set of seven rules to initially classify instances in the training data. Adding the knowledge of rules and an initial classification makes the system semi-supervised comparable to the gazetteer creation of Kozareva (2006) mentioned on page 16.

Because NERC is defined as the task of classifying instances into given named entity types, unsupervised models are rarely used for it.

To summarise "Learning and Classification" we can state that machine learning methods rely on two things: The quality and size of available training data and the set of used features. The availability of needed data underlies strong variations in different setups. There are many corpora dealing with news texts, some others exist for specialised fields like bioinformatics but annotated data generally lack in coverage of world knowledge while unlabelled data can not compete with their quality. For this reason the free encyclopedia Wikipedia received a great deal of attention of the NLP research community over the past years. It is semi-structured, has a great number of entries and is available in several languages, which makes it a useful source of information that may overcome the mentioned disadvantages of other resources. Therefore, the next chapter presents Wikipedia: its structural properties, how it can be accessed and which work related to this thesis already took advantage of it.

[16]The classes are undefined in respect to the fact, that the classifier does not know which type of instances is contained in the classes. It has no clue how the clusters could be labelled.

4 Wikipedia

Since the project started in 2001, Wikipedia (WP) has been known as the online encyclopedia created collaboratively by volunteers, which by now is available in 269 languages. Of these languages 29 contain more than 100,000 articles, 160 have more than 1000 entries[17]. The size, structure and free availability makes Wikipedia a useful source for a variety of research topics and was utilised for named entity recognition and disambiguation (Bunescu and Paşca, 2006), taxonomy induction (Ponzetto and Strube, 2007), question answering (Lita et al., 2004; Buscaldi and Rosso, 2006), ontology construction (Suchanek et al., 2007), information extraction and enhancing other resources like Cyc[18] (Medelyan and Legg, 2008) or WordNet[19] (Ruiz-Casado et al., 2005).

Most of the non-multilingual work was performed on the English Wikipedia (WP_{en}) because it is, with more than 3 million articles, by far the largest version. In comparison, the second biggest, WP_{de} (German), has close to one million articles, WP_{fr} (French) on position three has 870,725 and from there the number decreases fast as it can be seen in table 2.

4.1 Structure

The knowledge contained in Wikipedia is structured by two concepts: pages and links between them. There are different types of pages to distinguish between the kinds of information they provide. This structure is identical in every language version of Wikipedia. Wikipedia page titles are represented by small capitals.

4.1.1 Pages

Everything a user can see browsing through Wikipedia is a page. Every page lives within a namespace which denotes sections within Wikipedia and bears a unique title in that namespace. For example the article with the title HEIDELBERG lives in namespace *Main*, and its discussion page with the same title lives in the

[17]Information on statistics are taken from `http://stats.wikimedia.org/EN/Sitemap.htm` on Dec 18. 2009.

[18]Lenat (1995)

[19]Fellbaum (1998)

Language code	Language	Article Count
en	English	3,106,727
de	German	988,060
fr	French	870,725
pl	Polish	657,506
ja	Japanese	640,698
it	Italian	628,383
nl	Dutch	567,408
es	Spanish	558,041
pt	Portuguese	525,253
ru	Russian	453,608
sv	Swedish	337,963
zh	Chinese	274,402
no	Norwegian	238,657
fi	Finnish	223,110
ca	Catalan	207,655
uk	Ukrainian	180,893
hu	Hungarian	150,858
cs	Czech	143,960
tr	Turkish	142,299
ro	Romanian	134,476
eo	Esperanto	122,027
ko	Korean	121,394
da	Danish	119,304
vo	Volapük	118,781
id	Indonesian	115,889
sk	Slovak	111,136
vi	Vietnamese	106,982
ar	Arabic	106,225
sr	Serbian	104,207

Table 2: Wikipedias with more than 100,000 articles sorted by size

namespace *Talk*. This is a general concept of Wikipedia: there are pages in the basic namespaces and those in the talk namespaces to discuss them. Table 3 gives an overview.

The generally interesting pages for NLP tasks are in the basic namespaces *Main* and *Category*. If no other namespace is mentioned explicitly we are talking about namespace *Main* whose articles describe one single, unambiguous entity. Uniqueness of pages is useful property because entities occurring in free text often are ambiguous and may be disambiguated by searching for a matching Wikipedia article (Bunescu and Paşca, 2006).

Another property of a page is that it can be a redirect. Redirect pages have a title which is common for a referred article, but the article itself is named elsewise. For example WWW redirects to WORLD WIDE WEB. As redirects include abbreviations, nicknames and other aliases, they can help to find multiple surface referring to the same entity.

In case a term may describe several things, a disambiguation page is set up. E.g. the disambiguation page "WWW (DISAMBIGUATION)" informs that *WWW* also may refer to an early web browser called WORLDWIDEWEB, the theme park WILD WEST WORLD or amongst other suggestions WICKED WITCH OF THE WEST, a character from *The Wonderful Wizard of Oz*. Disambiguation pages can help to identify matching candidates for named entity disambiguation.

As every title must be unique, entities with the same name are distinguished by a descriptive word which is added in braces to the title. Often the most popular entity has the title without that description. For example MICHAEL JACKSON pictures the deceased American pop singer, but the disambiguation page MICHAEL JACKSON (DISAMBIGUATION) lists many people with a similar name, e.g. MICHAEL JACKSON (ACTOR), MICHAEL JACKSON (WRITER), MICHAEL JACKSON (RADIO COMMENTATOR), MICHAEL A. JACKSON (SHERIFF), . . . and MICHAEL J. JACKSON (ACTOR).

Basic Namespace		Talk Namespace	
0	Main	Talk	1
2	User	User talk	3
4	Wikipedia	Wikipdia talk	5
6	File	File talk	7
8	MediaWiki	MediaWiki talk	9
10	Template	Template talk	11
12	Help	Help talk	13
14	Category	Category talk	15
100	Portal	Portal talk	101

Table 3: Wikipedia namespaces and corresponding numerical values in the database.

4.1.2 Links

The organising elements of Wikipedia pages are links between them. They are just placed in the text of pages in Mediawiki[20] markup style. A link to another Wikipedia page in the same namespace is created when putting the title of the page in brackets. Thus writing [[Ramones]] in the page's source text results the rendered page to show a link with the surface form "Ramones" to the article of the punk band: `http://en.wikipedia.org/wiki/Ramones`.

In order to display another surface form, users can place so called piped links: [[Ramones|punk band]] still links to the same article, but shows "punk band" in the text. Because links point to pages they are unambiguous, too.

To link to another namespace users just have to prefix the desired namespace together with a colon. A link to the discussion page of the article "Ramones" looks like this: [[Talk:Ramones]]. Another example taken from the article Ramones is a link to the category "American punk rock groups" [[Category:American punk rock groups]]. A page linking to another page in namespace *Category* is considered to be in this category.

Although language links are not a namespace of Wikipedia, they follow the same syntax [[language code:Ramones]] where language code follows the abbreviations defined by ISO 639[21]. The result is a link to the article "Ramones"

[20]Mediawiki is the software Wikipedia is based on: `http://www.mediawiki.org/wiki/MediaWiki`

[21]`http://www.loc.gov/standards/iso639-2/php/code_list.php`

in another language version of Wikipedia. The article title may differ between languages, e.g. the catalan link is [[ca:The Ramones]]. Wikipedia editors are adding language links manually which means that every page with a language link provides a translation of the article for free.

All linking styles can be mixed.

4.2 Access to Wikipedia data

To solve a NLP task using Wikipedia, structured access to the content is needed. The Wikimedia Foundation[22] who runs the servers of Wikipedia provides downloadable database data in XML and SQL format on `http://download.wikimedia.org/`. The pages-articles.xml files contain the current versions of article content which means it is a snapshot of the database including every page in namespace *Main*. This database dump includes all links, but they are hidden in the unparsed article texts. This problem is overcome by the SQL files which are sorted by tables of the database[23] and allow to sort for pagelinks, languagelinks or categorylinks. After all, there are three ways to access the data:

1. Query the web page
2. Work on the XML dumps
3. Work on a database

As querying the web page is discouraged[24] and would be too slow for large computational work anyway, we leave out that option. Working on the XML file is fine for tasks that need to look at needed information only once because it can be huge and needs to be parsed, which makes accessing random pages in adequate time impossible. As this work needs dynamic access to articles and categories, an Advanced Programming Interface (API) to the database is needed. Torsten Zesch and Gurevych (2008) introduced the Java based Wikipedia API *JWPL* which suits

[22]`http://wikimediafoundation.org/wiki/Home`

[23]Visit `http://www.mediawiki.org/wiki/Manual:Database_layout` to learn about the database layout and available tables.

[24]`http://en.wikipedia.org/w/index.php?title=Wikipedia:Database_download&oldid=335585023#Please_do_not_use_a_web_crawler`

the task. It allows to access various kinds of information in an object oriented manner. The only important feature it is lacking is easy multilingual access which means a comprehensive way to follow language links. The initial idea of this thesis included the utilisation of categories across languages. Consequently, a new API with this functionality had to be written.

mwdb - a Python API for Wikipedia

The software *mwdb* is a Python API for Wikipedia (or Mediawikis in general) granting object oriented access to articles, links and categories across different language versions of Wikipedia. It was written together with Wolodja Wentland and can be found at his github account under `http://github.com/babilen/`[25].

Maybe the best way to talk about the functionality of mwdb is to present it in an interactive Python session. Everything following >>> is user input, lines without a prefix are program output and text following # is a comment to explain what is happening. Python keywords are highlighted in boldface. To increase readability some parts replaced by "`...` " were left out and some of the output was sorted. So let us have a look at the first steps using mwdb:

```
>>> import mwdb
#connect to a postgresql database and discover installed Wikipedia databases
>>> mwdb.databases.discover('postgresql','psycopg2', username,password, host)
#dictionary of found database objects of the different language versions
>>> mwdb.databases
Databases({
  u'ru': PostgreSQLDatabase(localhost, wp_bs_20091114),
  u'da': PostgreSQLDatabase(localhost, wp_da_20091114),
  u'de': PostgreSQLDatabase(localhost, wp_de_20091110),
  u'en': PostgreSQLDatabase(localhost, wp_en_20091103),
  u'fr': PostgreSQLDatabase(localhost, wp_fr_20091111),
  u'ja': PostgreSQLDatabase(localhost, wp_ja_20091117),
  u'nl': PostgreSQLDatabase(localhost, wp_nl_20091112)
  u'pl': PostgreSQLDatabase(localhost, wp_pl_20091116),
...
  u'zh': PostgreSQLDatabase(localhost, wp_zh_20091112)}

#access to English Wikipedia
>>> enwp = mwdb.Wikipedia('en')
```

[25]Also on this page: The automatic Wikipedia download tool *wp-download* and the database import tool *wp-import*.

```
#the Wikipedia object
>>> enwp
Wikipedia(en)

#show methods of enwp object
>>> dir(enwp)
[..., 'get_article', 'get_category', 'iter_articles',
  'iter_categories', 'language']

#load article page object with the title ``The Smurfs''
>>> smurfs = enwp.get_article(u'The_Smurfs')

#article objects are created for every language dynamically
>>> type(smurfs)
EN_Article(u'The_Smurfs')

#access to article text
>>> print smurf.raw_text
...
'''''The_Smurfs''''' (''Les Schtroumpfs [original French/Belgian name for
them]'') are a fictional group of small blue creatures who live in Smurf
Village somewhere in the woods. The [[Belgium|Belgian]] [[cartoonist]]
[[Peyo]] introduced Smurfs to the world in a series of [[comic strips]],
making their first appearance in the Belgian [[Franco-Belgian comics
magazines|comics magazine]] ''[[Spirou (magazine)|Spirou]]'' on October 23,
1958. ...

#access to articles linked by the article object
>>> print smurfs.article_links
[EN_PageLink(u'A_Yabba_Dabba_Doo_Celebration:_50_years_of_Hanna-Barbera'),
  EN_PageLink(u'Anders_Hedberg'), EN_PageLink(u'Andr\xe9__Franquin'),
  EN_PageLink(u'Animated'), EN_PageLink(u'Animated_television_series'), ...,
  EN_PageLink(u'Weekend_Today'), EN_PageLink(u'Yvan_Delporte')]

#looking for articles linking to this one results in a list of PageLink
#objects..
>>> smurfs.article_links_in[0]
EN_PageLink(u'The_Smurfs')

#..which we can ask for its source (the goal is 'The_Smurfs', obviously). The
#source page is an article object itself with the same methods as article page
#object ``smurfs''
>>> smurfs.article_links_in[0].source_page
EN_Article(u'Belgium')
```

Those are some basic functionalities of the article objects, more are available, but will not be presented here. Note that data which is directly accessible via a table in the database is realised as a property[26], while data needing the consultation of more tables or any computation is realised as a method usually starting with “get“ for elements or lists or ”iter“ for iterables. For example article.title is a property and article.iter_categories_startwith(start_string) is a method.

In order to keep article or page objects small, lazy loading is used to execute article.raw_text from the database, which means that it is only loaded when accessed. The continued interactive example shows access to categories and related methods.

```
#access to the categories of an article
>>> smurfs.categories
[..., , EN_Category(u'Fictional_characters_in_comics'),
  EN_Category(u'Fictional_dwarves'), EN_Category(u'Fictional_life_forms'),
  EN_Category(u'NBC_network_shows'), EN_Category(u'Smurfs')]


#getting category object EN_Category(u'Fictional_life_forms')
>>> fictional_life_forms = smurfs.categories[-3]

#A category is placed in categories as well..
>>> fictional_life_forms.categories
[EN_Category(u'Fictional'), EN_Category(u'Life'), EN_Category(u'Organisms')]

#.. and has subcategories
>>> fictional_life_forms.subcategories
[EN_Category(u'Discworld_peoples'), EN_Category(u'Marvel_Comics_species'),
  EN_Category(u'DC_Comics_species'), EN_Category(u'Star__Wars_races'),
  EN_Category(u'Fictional_animals'), EN_Category(u'Lists_of_fictional_species'),
  ...
  EN_Category(u'Pok\xe9mon_species'), EN_Category(u'Video_game_creatures')]
```

As we can see, walking through the category graph is very easy. Some other methods, like gathering several levels of subcategories were implemented in this thesis, but are not in mwdb yet.

By now you might have come across characters like *\xe9* in e.g. *EN_Category(u'Pok\xe9mon_species')*. They are just the Unicode representation of a string which is used internally. In Python, the *u* in front of a string means that it is a Unicode string. When printed, those characters are encoded and correctly display *EN_Category(Pokémon_species)*.

[26]Properties in Python are usually used to allow easy and comfortable access on variables of objects. They are accessed via object.**property_name** as opposed to object.**method_name()**

The last snippet of the interactive session gives insight into the multilingual possibilities of mwdb:

```
#show the stored language links of an article
#note that the method still operates on the English WP
>>> smurfs.language_links
[EN_LanguageLink(u'als', u'Les_Schtroumpfs'),
  EN_LanguageLink(u'ar', u'\u0627\u0644\u0633\u0646\u0627\u0641\u0631'),
  EN_LanguageLink(u'br', u'Schtroumpfed'),
  EN_LanguageLink(u'bs', u'\u0160trumpfovi'),
  EN_LanguageLink(u'de', u'Die_Schl\xfcmpfe'),
  EN_LanguageLink(u'es', u'Los_Pitufos'),
  EN_LanguageLink(u'fr', u'Les_Schtroumpfs'),
  EN_LanguageLink(u'ja', u'\u30b9\u30de\u30fc\u30d5'),
  EN_LanguageLink(u'nl', u'Smurf'),
  EN_LanguageLink(u'pl', u'Smerfy'),
  EN_LanguageLink(u'ru', u'\u0421\u043c\u0443\u0440\u0444\u044b'),
  ...
  EN_LanguageLink(u'zh', u'\u84dd\u7cbe\u7075')]

#follow the language links to the actual translated articles creating objects
#in other Wikipedia_language instances
>>> list(smurfs.iter_translations())
[ALS_Article(u'Les_Schtroumpfs'),
 AR_Article(u'\u0627\u0644\u0633\u0646\u0627\u0641\u0631'),
  BR_Article(u'Schtroumpfed'),
  BS_Article(u'\u0160trumpfovi'),
  DE_Article(u'Die_Schl\xfcmpfe'),
  EO_Article(u'Smurfo'),
  ES_Article(u'Los_Pitufos'),
  FR_Article(u'Les_Schtroumpfs'),
  JA_Article(u'\u30b9\u30de\u30fc\u30d5'),
  NL_Article(u'Smurf'),
  PL_Article(u'Smerfy'),
  RU_Article(u'\u0421\u043c\u0443\u0440\u0444\u044b'),
  ...
  ZH_Article(u'\u84dd\u7cbe\u7075')]
```

Every page object in Wikipedia, including articles and categories, which has language links, can load the according page object in another language if that language is installed. Connections to other databases are opened dynamically for this purpose. To our knowledge no other API provides multilingual access to Wikipedia in such an easy way.

4.3 Related work

As Wikipedia is a valuable resource, many researchers have used it for different tasks. This section presents some of the work related to this thesis. This can mean that it either had some interesting and inspiring thoughts and conclusions or it produced useful data.

In 2006 Razvan Bunsecu and Marius Paşca exploited the "high coverage and rich structure of the knowledge encoded in an online encyclopedia"(Bunescu and Paşca, 2006, page 1) for named entity disambiguation (NED). If an ambiguous string, a phrase that can refer to several unique entities, is found in a text, NED is the task of deciding which of the possible candidates is meant in this context. They took the setup of a web search as an example of where NED is useful: People searching for "*Python*" may refer to the snake, programming language or movie. A common name like "John Williams" is carried by several persons (composer, wrestler ...), so the goal is to be able to improve the effectiveness of a search by grouping the results by the different entities. They achieve it by using correlations between entities in Wikipedia, their categories and contexts they appear in to compute similarities between a query and its named entity candidates. Most interesting in this paper was the generation of:

- a collection of named entities,
- a disambiguation dictionary consisting of proper names as keys and named entities they may denote as values
- ambiguous contexts for different entities.

To detect all named entity articles $A_{NE} \in \mathrm{WP}_{en}$ Bunescu and Paşca (2006) developed a simple but effective heuristic for named entity detection based on capitalisation.

The disambiguation dictionary is constructed by use of redirect and disambiguation pages. It maps proper names to unique named entities they may refer to. The keys of the dictionary are referring expressions and the values are always named entities. For example the key JOHN WILLIAMS is mapped to JOHN WILLIAMS (COMPOSER), JOHN WILLIAMS (WRESTLER),... and JOHN WILLIAMS (VC).

This dictionary allows to search for ambiguous contexts for different named entities in Wikipedia. This was done by looking for links to a known named entity article a_{NE} with an ambiguous surface string within other articles. For example if an article contains a link "[[John Williams (wrestler)|John Williams]]" we know that John Williams (wrestler) is referred to. The disambiguation dictionary allows to look up other named entities, which "John Williams" may describe. By saving the contexts of these cases a useful collection of training material for named entity disambiguation is created.

This idea was extended by Wentland et al. (2008) who presented ***HeiNER*** – *the* ***Hei****delberg* ***N****amed* ***E****ntity* ***R****esource*. They realised that creating the disambiguation dictionary and the ambiguous contexts solely rely on Wikipedia's linking structure and therefore is language independent. Additionally because the context data can be used as training material they decided that it should not be restricted to ambiguous contexts, but include all occurrences. Consequently, they applied the NER heuristic of Bunescu and Paşca (2006) to the English Wikipedia and used interlanguage links to translate the named entities, which resulted in a collection of translated named entities A_{HeiNER} with more than 1.5 million named entity articles in WP_{en}. Then the disambiguation dictionaries and context datasets were created for a chosen set of 16 languages[27].

The evaluation of the NER heuristic showed a precision of 95% for the articles in A_{HeiNER} after the CoNLL annotation guidelines and the context datasets provide a median between four and seven contexts per named entity[28]. HeiNER's size and multilingualism makes it a valuable resource for tasks like named entity recognition, disambiguation, translation and transliteration. Yet, it is not applicable for NERC because it is missing named entity type labels. This problem will be taken care of in the next chapter.

[27]The method allows the creation in all languages available in Wikipedia

[28]With the exception of Norwegian which had a median of two contexts per named entity

5 Classification of HeiNER's named entities

The advantage of HeiNER's data is that its collection of named entities is large and there is a beneficial number of examples per entry. The downside is that nothing is known about the named entities except for that they are named entities.

Common NERC systems rely on the size and quality of training data to learn correct classifications of named entity types to be able to estimate the type of a candidate in a given context. As HeiNER's size and quality was shown, eliminating its lack of named entity type assignments would increase its value significantly. Thus, the goal of this research is to classify the named entities contained in HeiNER with the help of Wikipedia's category system abiding by the CoNLL annotation guidelines shown on page 7 in order to create a multilingual training resource for NERC. The CoNLL guidelines were chosen because the evaluation of HeiNER's named entities was done following them and reported a precision of 95% which could not be guaranteed for other definitions. As the named entities and their translations are known, it is sufficient to classify them in English to apply the result to other languages covered by HeiNER as well.

Assigning NE types to articles is done in two steps. First, we collect well-defined categories for named entity types and use them to gather an initial list of classified named entities. Second, a bootstrapping approach is used to classify the remaining articles. The idea is described in subsection 5.1 and carried out in subsection 5.2 followed by its evaluation in 5.3.

5.1 Mapping Wikipedia categories to named entity types

Previous work has shown that the category system of Wikipedia offers useful information which helps to collect and represent world knowledge. The categorisation system of Wikipedia articles is very detailed and organised well. It should be possible to classify named entity instances with its help.

5.1.1 Initialisation

The first step is to find mappings between the CoNLL annotation guidelines and Wikipedia categories. For the named entity type *Person* there exists a similar category PEOPLE, *Locations* are found under PLACE NAMES and *Organisations* can be mapped to the category ORGANIZATIONS. *Miscellaneous* is more complex and therefore covered by more than one category. So what can we do with these coarse mapping on the way to classify the named entities in HeiNER? First, we need some definitions:

Definition. *With the set of named entity types $T = \{ P, L, O, M \}$ we distinguish Person, Location, Organisation and Miscellaneous.*
By C we denote the set of all categories in the English Wikipedia.
Subcategories are given as a relation $S \subset C \times C$ with $(x, y) \in S$ if and only if x is a subcategory of y.
$A_H \subset A$ are all English articles in HeiNER.
Wikipedia provides a relation $R \subset A \times C$ that assigns articles to a set of categories they are placed in.

The goal of this work is to find a mapping $f : A \to T$ that assigns a type to each article. In essence, our approach consists of two steps. First, we determine a set of categories for each type. Then, these are used to initialise a bootstrapping algorithm that retrieves types for the articles.

We start with the set C_t of categories that belong to type t. We assume that for each type $t \in T$ and every relation $(a, c) \in R$ it can be deduced that $c \in C_t \Rightarrow f(a) = t$. In other words, every article that can be found in a category of type t can be considered to belong to this type, too. If we thought of a way to find categories with their denoted type, we would just have to look up the articles placed in them to classify articles and subsequently create the desired mapping. Because there are too many categories to build all category sets by hand, a semi-automatic approach is needed.

As categories can have high number of subcategories, the idea occurs that a set of seed categories denoted to a type may induce more categories for this type by following subcategory links recursively. This could be done for every type t like this:

Collect a set of seed categories C_seed_t and assemble all related categories recursively. For this, we need all subcategories k with $(k, c) \in S$ for some $c \in$

C_seed_t. We add all these subcategories to the initial seeds and repeat the process until no more new categories are found. The result is the set C_t of categories that are related to t.

The seed categories can be identified manually and need to be of high quality. On the one hand, the seeds have to be broad enough to be able to collect as many articles as possible, on the other hand, they have to be selected properly, otherwise every step in this chain could add elements not relevant to t. Section 5.2.1 will deal with the details of collecting seeds. However, no matter how good the quality of the manually selected seed categories is, it is unlikely for the derived C_t to comply with the definition that all categories dealing with t are covered. There may exist categories that can not be obtained by following subcategory links of the seed list. Therefore the collected type-categories are not sufficient to classify all articles in HeiNER by looking them up.

As much as the C_t are not perfect, in practice the lookup can still generate an initial list of classified articles $A_{initial\_t}$. This initialisation step leaves the set of unclassified articles, $A_{initial\_unclassified} = A_H - A_{initial\_t}$, so the new challenge is to find a way to identify the types for the remaining unclassified articles starting from $A_{initial\_t}$. For that, we have to ask what new useful information $A_{initial\_t}$ contains.

5.1.2 Bootstrapping

All articles in the initial list of classified articles are inferred from the type-categories C_t. Nevertheless, the inferred articles are placed in additional categories that can not be reached from the seed list. By taking a look at all the categories the articles in $A_{initial\_t}$ are placed in, we can collect these new categories. They may help to identify the types of the left-over articles that are yet unclassified as it was impossible to infer an appropriate type from C_seed_t. They can be seen as a footprint or *signature* of a named entity type aligning concepts that are shared between articles and moreover adding abstract knowledge to the concrete entities.

A named entity type can be represented by this category signature as a category vector $\vec{v}_t$. The vector's length is the number of distinct categories found by looking up the categories that members of $A_{initial\_t}$ are placed in. The values are calculated by counting all of the categories' occurrences. The point of this vector is, that it

```
Castle_Bromwich_Assembly
Daimler_Armoured_Car
Daimler_Conquest
Daimler_Consort
...
Daimler_Sovereign
Jaguar_Advanced_Lightweight_Coupe_Concept
Jaguar_C-Type
Jaguar_Cars
Jaguar_D-Type
...
Jaguar_XKSS
Lanchester_Motor_Company
Owen_Sedanca
```

Figure 2: Excerpt of articles in A_{jaguar}

can be used to classify entities in $A_{initial\_unclassified}$ by computing its similarity to the unclassified articles. A detailed description follows in section 5.2.3, but first, step back and have a look at an example of how the set of categories is generated:

Consider *Jaguar*[29] to be a named entity type for demonstrating purposes. For this type the initial seed c_seed_{jaguar} is the category JAGUAR which with all subcategories results in C_{jaguar}:
{JAGUAR, DAIMLER, DAIMLER VEHICLES, JAGUAR VEHICLES and JAGUAR FORMULA ONE CARS}.

Looking up the articles placed in the categories in C_{jaguar} results in a set of 60 classified articles $A_{initial\_jaguar}$. Some of them can be seen in figure 2 (page 32). The articles mainly list car types of the manufacturers *Jaguar* and *Daimler Motor Company*[30]. In this example the assumption that the articles placed in C_{jaguar} deal with the type *Jaguar* as well seems to be correct: LANCHESTER_MOTOR_COMPANY was a car manufacturer that merged with Jaguar, CASTLE_BROMWICH_ASSEMBLY is a car factory owned by Jaguar and OWEN_SEDANCA is a car type.

After that, the vector $\vec{v}_{jaguar}$ is created by the algorithm presented in figure 3 with the help of the categories that the classified articles $A_{initial\_jaguar}$ are placed

[29] Jaguar denotes the car manufacturer, not the animal.
[30] A company owned by Jaguar, which should not to be mistaken for Daimler-Benz.

in. We step through every article and look at its categories. If a category is not in $\vec{v}_{jaguar}$ yet, we add it with a value of one, else we increment the current count by one. The resulting category vector illustrated in figure 4 (page 34) consists of 135 distinct categories adding up to a sum of 336 mentions in $A_{initial\_jaguar}$.

```
def compute_vector($A_{initial\_t}$):
  category_vector = {} # $\vec{v}t$ stored as a dictionary
  for $a$ in $A_{initial\_t}$:
    for $c$ in $a.catgories$:
      if category_vector.has_key($c$):
        category_vector[$c$] += 1
      else:
        category_vector[$c$] = 1
  return category_vector
```

Figure 3: Python-Pseudocode algorithm of a function to build the category vector. The vector is stored in a dictionary where the category name is the key and the count its value.

Most categories are in a hypernym relation to their articles. Especially the more frequent ones like JAGUAR_VEHICLES, DAIMLER_VEHICLES and SPORTS_CARS might be used as ontological concepts. Other categories help to organise the articles by specific criteria like time (1960S_AUTOMOBILES, 1950S_AUTOMOBILES, VEHICLES_INTRODUCED_IN_2006, ...). ALL_ARTICLES_WITH_UNSOURCED_STATEMENTS and ALL_ORPHANED_ARTICLES are exemplary for Wikipedia maintenance categories. They appear frequently and are set often automatically by bots to flag articles which need attention by Wikipedia editors to improve their quality. As they are widespread they do not tend to be more significant for one topic over another.

Let's recapitulate what happened up to this point: In order to create named entity type vectors $\vec{v}_t$, the processing chain starts with manually selected seed categories C_seed_t to create C_t via all of the seed's subcategories. After that, it collects the articles $A_{initial\_t}$ by looking them up in C_t. Examining the categories of $A_{initial\_t}$ and counting their occurrences results in the type vectors $\vec{v}_t$.

The type of unclassified articles can now be identified by computing the sim-

```
{Jaguar_vehicles:38,
Rear_wheel_drive_vehicles:24,
Daimler_vehicles:13,
Sports_cars:13,
1960s_automobiles:11,
1950s_automobiles:9,
2000s_automobiles:8,
All_articles_with_unsourced_statements:7,
Sedans:7,
Rear_mid-engine,_rear-wheel_drive_vehicles:5,
...,
All_orphaned_articles:1
Vehicles_introduced_in_2006:1}
```

Figure 4: $\vec{v}_{jaguar}$ resulting from $A_{jaguar}.categories$

ilarity between the type-vectors $\vec{v}_t$ and the articles' categories and assigning the best matching type to them. This allows a bootstrapping approach to classify the yet unclassified articles: We use ten iterations. In each iteration the similarity of articles and the four NE type-vectors is computed and the type with the highest similarity score is assigned to the corresponding articles. The resulting list is sorted descending by similarity scores and the ten percent of unclassified articles with the highest scores are added to their matching named entity types and moved from the list of unclassified articles to the classified ones. The categories of these new classified articles are counted and added to the vectors of their respective NE type. After that, we go to the next iteration step using the updated category vectors for new similarity computations. The complete algorithm is described in Python-Pseudocode on page 35.

By now we explored the idea how to carry out the classification of the named entities in HeiNER. The next section will present the work on real data.

```
$A_{classified} = A_{initial}$
$A_{unclassified} = A_{initial\_unclassified}$
$n$ = 10 #iteration count
$c\_per\_i$ = len($A_{unclassified}$)/$n$ #classifications per iteration
$\vec{V}_T = [\vec{v}_{person}, \vec{v}_{organisation}, \vec{v}_{location}, \vec{v}_{miscellaneous}]$

for $k$ in range($n$):
    $A_{new\_classified}$ = []
    for $a$ in $A_{unclassified}$:
        $candidate$ = best_similarity_tuple($a$, $\vec{V}_T$) #see function below
        $A_{new\_classified}$.append($candidate$)
    #sort descending by similarity (tuple index 2)
    $A_{new\_classified\_sorted}$ = sort_desc($A_{new\_classified}$, key=2)

    #iterate over c_per_i(10%) of the highest rated articles
    for $a_{new\_classified}$ in $A_{new\_classified\_sorted}$[:c_per_i]:
        $article$ = $a_{new\_classified}$[0]
        if $article$ == 'unclassified':
            break #no more classified articles left in $A_{new\_classified\_sorted}$
        $A_{classified}$.append($article$)
        $A_{unclassified}$.remove($article$)
        $type$ = $a_{new\_classified}$[1]
        $\vec{V}_{type}$ = $\vec{V}_{type}$.add_categories_of_article($a_{new\_classified}$[0])

def best_similarity_tuple($a$, $\vec{V}_T$):
    $maximum$ = 0
    $best\_type$ = 'unclassified'
    for $vector$ in $\vec{V}_T$:
        $similarity$ = compute_similarity($a.categories$, $vector$)
        if $similarity$ > $maximum$:
            $maximum$ = $similarity$
            $best\_type$ = $vector$.get_type_name()
    #return tuple with the result
    return ($a$, $best\_type$, $maximum$)
```

5.2 Classification of named entities contained in HeiNER

By now the idea of how to approach the classification of HeiNER's named entities is given, so it is time to present the work with real data. These are taken from the English Wikipedia of November 3rd 2009. First, the NER of HeiNER was run to generate an updated collection of named entities in Wikipedia$_{en}$. A total of 2,225,193 were found[31]. Afterwards, 721 randomly chosen instances were manually annotated and set aside to create an evaluation set (cf. Evaluation, section 5.3), which left $2,224,472$ in A_{HeinER}. The next step is the collection of seed categories.

5.2.1 Collecting the seed categories

The seed categories were gathered by looking at several randomly chosen named entity pages of every named entity type and travelling up the category tree until the most abstract but as concrete as necessary category was found matching the descriptions of CoNLL annotation guidelines (cf. table 1 on page 7). For example to find seed categories for the type *Person* we start looking at the article of Barack Obama and examine its categories, where we find LIVING PEOPLE amongst many others. As it is the most abstract and best fitting category we examine the categories LIVING PEOPLE is placed in and continue with the selection leading to the following chain:

BARACK OBAMA ⇒ LIVING PEOPLE ⇒ PEOPLE BY STATUS ⇒ PEOPLE

The next higher category would be HUMANS but as it contains other categories covering a wide topic range like HUMAN PHYSIOLOGY or HUMAN RIGHTS it is not a useful choice for the named entity type *Person*. The same question has to be asked for the category PEOPLE: Do the subcategories introduce topics which do not fit the named entity type? If there are categories which do not apply, throw them away and take all the remaining subcategories at this level in order to avoid noisy data. Those are some of the removed categories in PEOPLE:

[31]The original work had 1,547,586 entries (Wentland et al., 2008). The difference is solely based on the growth of Wikipedia$_{en}$.

- Biography – deals with biographical books
- List of People – contains mostly 'List of'-articles which are not named entities
- Categories named after actors – often leads to work of actors like their films
- Personal timelines – only has articles about the timelines, not persons themselves

In the end most of the remaining categories for *Person* had the form People by XY, in which XY is an organisational topic like "occupation" or "religion". There are 15 categories in C_seed_{person}. This manual work was done for every named entity type. Finding the categories for other types was not always as clear as for *Person* because some types have more than one suitable supercategory to start from and thus have a broader set of seed categories taken into account. So $C\_seed_{organisation}$ ends up having 75 and $C\_seed_{miscellaneous}$ 27 categories while $C\_seed_{location}$ has 15. The complete categories found for the CoNLL classes are listed in the Appendix on page 63 ff.

5.2.2 Initialising the named entity types

Starting with C_seed_t we can build C_t by computing all the subcategories of C_seed_t as described on page 30. So we try to do that for C_seed_{person} and find C_{person} to have more than 526,000 categories. The same result occurs with $t =$ *Location*. Checking WP_{en} reveals that it has a total of 528,128 categories and all of them were added to the respective C_t, which makes the collection useless. The problem is that by taking all subcategories of the categories in C_seed_t we treat the category system as a tree although it is a graph. The removal of seemingly noisy categories was not sufficient to avoid going off the track and adding categories which introduce unrelated parts of the category graph.

An easy way to overcome this problem is to restrict the depth of the search for subcategories. The depth should be chosen deep enough to maximise the ensuing collection of $A_{initial\_t}$, but also avoid the introduction of wrong categories.

Examining the category system leads to the conclusion that a depth of two is sufficient to reach most of the wanted categories dealing with t and flat enough to exclude unrelated categories. Articles that could not be reached because of the restriction of search depth are expected to be added later in the bootstrapping part if they are related to a named entity type.

The algorithm in figure 5 shows how the subcategories are computed with a depth restriction of two.

```
$C_t = C\_seed_t$
for i in range(2):
  $C\_subcat_t$ = set()
  for $c_t \in C_t$:
    $C\_subcat_t = C\_subcat_t.union(c_t.subcategories)$
  $C_t = C_t.union(C\_subcat_t)$
return $C_t$
```

Figure 5: Python-style pseudocode to build C_t with search depth = 2

As explained earlier, the initial set of classified articles is computed by looking up articles in C_t. But not every found Wikipedia article should be added to $A_{initial\_t}$. Recall that $A_{initial\_t}$ should only contain articles of HeiNER so that the signature of categories only consists of those assigned to named entity articles and ignores articles about common words to improve the signature's accuracy. Articles about common words may introduce unwanted categories, e. g. the category CHOIRS is used as a seed for *Organisations*. The article SHOW CHOIR[32] is placed in CHOIRS and is not a named entity but a common word. Furthermore, the article is placed in the category DANCE. Because dancers like ALFREDO CORVINO[33] are placed in DANCE, too, the category is associated with the named entity type *Person* as well, which is a source for erroneous mixing of named entity types. We do not want ALFREDO CORVINO to be associated with *Organisation*, so we avoid adding erroneous common word entries by adding articles to $A_{initial\_t}$ that are also contained in HeiNER.

[32]http://en.wikipedia.org/w/index.php?title=Show_choir&oldid=336334892
[33]http://en.wikipedia.org/w/index.php?title=Alfredo_Corvino&oldid=335208937

t	C_seed_t	$C\_seed_t.$ $subcategories$	C_t	$A_{initial\_t}$	$\frac{A_{initial\_t}}{C_t}$
Person	15	9,625	9,640	513,821	54.47
Location	15	2,783	2,798	45,000	16.08
Organisation	75	8,033	8,108	140,708	17.35
Miscellaneous	27	4,747	4,774	52,942	11.09

Table 4: Counts of categories and articles for a named entity type t derived from seed categories. Double classifications are included.

Table 4 shows the results of the creation of $A_{initial\_t}$ starting from C_seed_t. We can see that the number of seed categories is not necessarily important to reach many subcategories: *Person* and *Location* have the same count of seed categories, 15, but C_{person} is almost 3.5 times bigger than $C_{location}$ and has about 1,500 categories more than $C_{organisation}$ which started with 75 seed categories. A conclusion would be that persons have a very fine grained categorisation. $C_{miscellaneous}$ remains in between the others with 4,747.

By far the most articles are found for *Person*. The motivation for this cannot solely be based on the superior count of C_{person} because $C_{organisation}$ is not that far behind, though $A_{initial\_organisation}$ is about 3.6 times smaller than $A_{initial\_person}$. $\frac{A_{initial\_t}}{C_t}$ can be seen as the productivity of a category and tells that C_{person} is about 5 times more productive than $C_{miscellaneous}$. In other words, most of Wikipedia's contributers write articles about named entities of the type *Person* and categorise them studiously.

Of the 2,224,472 articles in A_H 720,032 are classified in the initialising step, leaving 1,504,440 in $A_{unclassified}$. You might wonder why the numbers in row "$A_{initial\_t}$" of table 4 sum up to 752,471 and not 720,032. That is because some of the articles are found in more than one C_t and end up classified twice or even more. All of these ambiguous articles are removed and left unclassified in order to avoid errors because we do not allow multiple types for one entry and cannot easily decide which one of the assigned types is correct. This step leaves 502,173 entries in *Person*, 41,539 in *Location*, 128,433 in *Organisation* and 47,887 in *Miscellaneous*. Table 5 presents how many overlapping articles per named entity type pair occurred.

NE type	*Person*	*Location*	*Organisation*	*Miscellaneous*
Person	X	768	8,371	2,870
Location	768	X	2,310	544
Organisation	8,371	2,310	X	1984
Miscellaneous	2,870	544	1984	X

Table 5: Combinations of the named entity types. The numbers describe the count of articles which were assigned to both types.

Most of the double classified articles show up between *Person* and *Organisation*. One reason is that they hold most of the articles. Another one could be that many people are closely connected to organisations (founders, important employees . . .) so that they are placed into categories also belonging to the organisation. For example BILL GATES is in the categories MICROSOFT HISTORY and BILL & MELINDA GATES FOUNDATION PEOPLE which can be reached from the seed categories COMPANIES BY COUNTRY or FOUNDATIONS. But it is not safe to say that the double categorisations tend to be of type *Person* for this combination because this also happens the other way around when an organisation stands for people's names. For example SANTANA (BAND) is about the band of Carlos Santana and not the person, although it is placed in the category PEOPLE ASSOCIATED WITH THE HIPPIE MOVEMENT which can be looked up by following the seed category PEOPLE BY CENTURY.

The pair of *Person* and *Location* tends to contain people who are associated with a location, e.g. the city founder PRINCE CARL OF SOLMS-BRAUNFELS. Less frequent are locations associated with people like SLAUGHTERBRIDGE, a settlement in north Cornwall. The same tendency seems to apply to the pair with the second most overlaps, *Person* and *Miscellaneous*, where people dealing with one of the wide topics related to type *Miscellaneous* are listed for both. Those may be authors, martyrs, veterans or athletes (cf. the CoNLL guidelines on page 7 to recall the various topics of *Miscellaneous*).

Articles classified as *Organisation* and *Location* often are both: universities, schools, libraries, prisons, capitals and cities can appear as organisations/government bodies, as well as locations. They are disregarded though, because of the restriction that only one classification is allowed per named entity. The overlap between *Location* and *Miscellaneous* seems to contain many book titles that deal with locations (A JOURNEY TO THE CENTER OF THE EARTH, THE RESTAURANT AT THE END OF THE UNIVERSE, ...) or articles referring to locations with points of contact to *Miscellaneous* like MOUNT MERU (MYTHOLOGY). The last combination, *Organisation* and *Miscellaneous*, does not seem to follow any pattern.

While articles which were classified to be of three types were not analysed, two articles stood out as they were found in every named entity type's collection: GAKHAR HINDUS, a "Punjabi community living in India with an ancient recorded history"[34] and ROBERT OWEN, "a social reformer and one of the founders of socialism and the cooperative movement"[35]. The article about the *Gakhars* deals with all topics which allow to classify it by all named entity types. The respective categories are INDIAN FAMILY NAMES (*Person*), KHATRI CLANS (*Organisation*), JHANGOCHI DIALECT SPEAKING AREAS (*Location*) and HINDU COMMUNITIES[36] (*Miscellaneous*). *Robert Owen* should only be classified as *Person*. The categories allowing him to enter the wrong types eventually are WELSH SOCIALISTS (*Location*), FOUNDERS OF UTOPIAN COMMUNITIES (*Organisations*) and SPIRITUALISM (*Miscellaneous*).

The bottom line is, that the sorted out articles indicate that the general type classification is correct, although the categorisation is not always easy. On this first glance it looks like the approach of following the categories is promising. But this was just $A_{initial\_t}$; the next subsection shows the classification of the remaining collection of unclassified articles.

5.2.3 Classification via bootstrapping

Starting from the 720,032 classified articles in $A_{initial\_t}$, we use the algorithm as described in section 5.1.2 (see figure 3 on page 33) to construct a vector $\vec{v}_t$ for every named entity type. Table 6 shows the results. Row "$A_{initial\_t}$" lists the initialised articles without the double classified entries (cf. section 5.2.2), the third row presents the dimensions of the vector which are equal to the count of

[34]`http://en.wikipedia.org/w/index.php?title=Gakhar_Hindus&oldid=330345945`

[35]`http://en.wikipedia.org/w/index.php?title=Robert_Owen&oldid=332973163`

[36]Because it is eventually leading to the seed category SPIRITUAL THEORIES via HINDUISM

unique categories in $A_{initial\_t}$. Sum($\vec{v}_t$) is the addition of all categories found for that type, or in other words $\sum_0^n \vec{v}_{t_n}$. The numbers support the statement that persons seem to be classified very fine-grained with an average count of almost eight categories per article in that class while the articles of other NE types have between four and five categories on average.

NE type	$A_{initial\_t}$	dimensions of $\vec{v}_t$ (unique categories)	$sum(\vec{v}_t)$	$\frac{sum(\vec{v}_t)}{A_{initial\_t}}$
PER	502,173	132,098	4,037,634	7.86
LOC	41,539	35,880	228,468	5.08
ORG	128,433	72,184	694,523	4.94
MISC	47,887	33,110	229,438	4.33

Table 6: NE type vectors constructed from $A_{initial\_t}$ (without double classifications).

The vectors enable us to start the bootstrapping with the algorithm of section 5.1.2. It relies on a similarity measure between the initialised vectors and an article. The latter forms a comparable vector by taking the dimensions of the named entity type vector and putting the value one at the positions that correspond to the categories of the article. Other positions are filled with zeros which leaves a binary article vector. It follows, that we will always compare a binary article vector to a weighted type vector. That means, that the choice of the similarity measure has a big influence on the results. Thus, two different similarity measures are used in two setups to compute the similarity between the type vectors and categories of an article:

1. Cosine similarity
2. Dice's coefficient

The cosine similarity between two vectors x and y is defined as follows:

$$cosine(\vec{x}, \vec{y}) = \frac{\sum_{k=1}^{n} x_k y_k}{\sqrt{\sum_{k=1}^{n} x_k^2} \cdot \sqrt{\sum_{k=1}^{n} y_k^2}}$$

It computes the angle between the two vectors. That means, that only the directions of the vectors are taken into account and not their length. Because no negative examples exist the resulting similarity ranges between zero and one.

Similarities in terms of the Dice's coefficient is computed like this:

$$dice(\vec{x}, \vec{y}) = \frac{2 \cdot \sum_{k=1}^{n}(weight_{xk} \cdot weight_{yk})}{\sum_{k=1}^{n} weight_{xk} + \sum_{k=1}^{n} weight_{yk}}$$

The Dice's coefficient includes the count of shared elements in relation to all elements that are not zero. It considers the weights of the vectors by multiplying the shared elements[37]. The factor 2 keeps the result range between zero and one. Dice's coefficient was chosen over the similar Jaccard coefficient, because the latter punishes a small amount of shared non-zero entries which are likely in this scenario.

The described bootstrapping is done for each similarity measure starting from the initialised type vectors. Ten percent of the unclassified articles with the highest similarity scores are added to their respective class in every step, which includes an update of the type vectors before the next step starts. Because only articles with any categories can be classified by this method, 7,033 articles that are not placed in a category were removed from the unclassified articles leaving 1,497,407 to be classified.[38] Table 7 on page 44 shows the numbers of added articles per class in every step.

The first row show the runs or steps in the bootstrapping iterations. Rows two to five show the counts of new classified named entity articles. The highest amount of added articles is marked boldface in each line. "UNCL" presents the number of articles which are left to be classified. It decreases by 10% (=149,740) of the initially unclassified articles in each iteration. The last row, "no similarity", contains the number of articles where no class was found with a similarity value greater than zero.

The first bootstrapping step of both similarity measures starts with the classification of many locations. This indicates that the seed categories missed many of them, but the introduced vector for that type allows the automatic classification to fix that problem. Subsequent iterations show that there is no general bias towards locations which supports this analysis.

[37]As we multiply with a binary vector we just decide whether to add the value of the non-binary vector at that position or not.

[38]The story of this number in short: we started with 2,224,472 unclassified articles found by the heuristics, removed 720,032 because they were classified in the initialisation which results in 1,504,440. Minus the 7,033 articles without any categories leaves 1,497,407 articles to be classified.

run	PER	LOC	ORG	MISC	UNCL	no similarity
setup	0	0	0	0	2,217,439	–
initial	**502,173**	41,539	128,433	47,887	1,497,407	–
			Cosine			
1	3,999	**120,641**	23,469	1,631	1,347,667	268,979
2	1,216	11,456	42,997	**94,071**	1,197,927	138,481
3	1,414	**56,725**	38,220	53,381	1,048,187	55,302
4	33,664	11,763	39,064	**65,249**	898,447	40,133
5	50,990	10,690	17,511	**70,549**	748,707	35,007
6	44,166	24,131	22,569	**58,874**	598,967	30,329
7	14,924	39,565	33,347	**61,904**	449,227	26,988
8	4,482	45,417	37,201	**62,640**	299,487	24,531
9	3,392	38,138	38,711	**69,499**	149,747	22,322
10	4,057	26,395	38,719	**60,913**	19,663	19,663
Bootstrap	162,304	384,921	331,808	**598,711**	–	–
Total	**664,477**	426,460	460,241	646,598	–	–
Plus	32%	927%	258%	**1250%**	–	–
			Dice's coefficient			
1	5,271	**137,051**	6,406	1,012	1,347,667	268,979
2	17	25	**138,578**	11,120	1,197,927	154,585
3	1,266	58,780	**65,593**	24,101	1,048,187	61,319
4	36,595	16,952	**56,017**	40,176	898,447	40,755
5	**67,975**	31,508	25,819	24,438	748,707	35,919
6	38,196	**56,745**	45,219	9,580	598,967	31,076
7	16,166	**67,458**	54,813	11,303	449,227	28,301
8	8,969	**67,890**	52,944	19,937	299,487	24,941
9	5,581	**65,655**	46,860	31,644	149,747	22,337
10	5,751	41,301	**56,864**	26,323	19,508	19,508
Bootstrap	185,787	543,365	**549,113**	199,634	–	–
Total	**687,960**	584,904	677,546	247,521	–	–
Plus	37%	**1,308%**	427%	417%	–	–

Table 7: Bootstrapping with the similarity measures *cosine* and *Dice's coefficient*. The percentages in the **Plus**-lines are rounded.

Cosine similarity seems to be biased towards *Miscellaneous* because on average about 60,000 articles are added to this type per iteration. In eight out of ten times the lion's share of the new classified articles is added to it. The reason for that could be, that cosine similarity depends on the angle between two vectors. That means, a type vector that shares many categories with an article would be preferred over a vector sharing less but possibly higher weighted categories. *Miscellaneous* might have thematically wide spread categories supporting that effect. Nevertheless, the bias towards that type can not solely be based on this property, because the initialised vector is the one with the least dimensions in comparison to the others (cf. table 6 on page 42).

Bootstrapping using the Dice's coefficient tends to be biased towards *Location* and *Organisation*, the former showing an overall gain of 1,308 percent[39] (543,365 articles). In four of the iterations *Organisations* wins the majority of new classified articles, *Locations* is in advantage in five of the iterations leaving *Person* one major gain in the fifth run. Because Dice's coefficient takes the counts of categories into account, it is likely that the unclassified articles are placed in some of the categories that have a high count for *Locations* and *Organisations*.

The count of articles added to *Person* develops remarkably similar for both measures. They start with few new articles in the first three iterations, rise to many more additions in steps four, five and six to slow down again in the left iterations. In both cases eventually *Person* is the named entity type with the least added articles (cf. lines "Bootstrap"), but still the biggest count when summing it up with the initial count (cf. lines "Total"). No other named entity type shows such a strong correlation between the two different similarity measures. This indicates that most of the articles were already classified in the initialisation proving the seed categories for that type to be of high quality.

The row "no similarity" informs how many of the unclassified articles are not found similar to any of the named entity type vectors. It develops likewise for cosine similarity and Dice's coefficient. The steadily decreasing count proves an increase of coverage of the type vectors. One would expect a decrease in precision there as well. The counts of left unclassified articles – 19,663 for cosine and 19,508

[39]This growth is narrowed a little bit by the fact that it started with the smallest count of articles (41,539).

for Dice – show that there is a collection of articles placed in categories that are not shared by any of the classified articles. Their slight difference is explained by different type vectors in iteration nine. These vectors are used for the classification of articles in iteration ten. Thus, the vectors in the setup with Dice's coefficient contain categories of articles that were not yet added to the vectors of iteration nine in the cosine setup.

In conclusion, both bootstrapping versions are able to classify almost all of the unclassified named entities, but differ a lot in their results with the exception of the type *Person*.

5.3 Evaluation

As mentioned earlier at the beginning of section 5.2 (page 36 ff.), there were 721 randomly chosen instances of HeiNER$_{en}$ manually annotated and set aside to create an evaluation set. They include 295 instances of type *Person*, 192 of *Location*, 122 of *Miscellaneous* and 110 articles describing an *Organisation*. To measure the quality of the method, we classify the instances in the evaluation set by computing the similarity between them and the type vectors and assign the most similar type to the instance just as it would have been in the bootstrapping process. First, a closer look is taken at the results from the vectors that were generated by the initialisation step. After that, the bootstrapping is analysed in the same way.

5.3.1 Initialisation

The result is shown as a confusion matrix for each similarity measures in table 8. The rate of correct classifications varies from 35.25% (*Miscellaneous*, Dice's coefficient) to 81.02% (*Person*, Dice's coefficient). It is not surprising that *Person* is the best performing named entity type when we remember the earlier statement that articles of that type are categorised with high detail and that this named entity type has by far the highest count of instances after the initialisation (cf. page 39). This is underlined by the fact that almost no instances were classified incorrectly as a person in the other evaluation sets. Consequently, there is no much confusion between persons and other named entity types.

Eval. set	**PER**	**LOC**	**ORG**	**MISC**	**UNCL**
			Cosine		
PER (295)	232 (78.64%)	17 (5.76%)	25 (8.47%)	19 (6.44%)	2 (0.68%)
LOC (192)	0 (0%)	116 (60.42%)	21 (10.94%)	14 (7.29%)	41 (21.35%)
ORG (110)	1 (0.91%)	17 (15,45%)	**74 (67.27%)**	9 (8.18%)	9 (8.18%)
MISC (122)	1 (0.82%)	10 (8.2%)	**47 (38.52%)**	46 (37.7%)	18 (14.75%)
			Dice's coefficient		
PER (295)	**239 (81.02%)**	18 (6.1%)	23 (7.8%)	13 (4.41%)	2 (0.68%)
LOC (192)	0 (0.0%)	**123 (64.06%)**	19 (9.9%)	9 (4.69%)	41 (21.35%)
ORG (110)	2 (1.82%)	21 (19.09%)	71 (64.55%)	7 (6.36%)	9 (8.18%)
MISC (122)	4 (3.28%)	12 (9.84%)	45 (36.89%)	43 (35.25%)	18 (14.75%)

Table 8: Confusion matrix for the CoNLL named entity types. Members of evaluation sets for every type were classified by computing similarities to the initialised named entity type vectors. The overall highest values (cosine and Dice similarity) are marked as boldface. The numbers in braces show the fraction of the absolute numbers that are given in the first row.

Considering that 21.35% of the articles were left unclassified, only 18.23% (cosine) and 14.59% (Dice) of the locations were explicitly classified wrong. Unclassified articles art those, where none of the instances in the evaluation set *Location* had categories that could be found in any of the named entity type vectors. This could either mean that the seed categories for this type were not chosen broad enough or that articles of type *Location* are placed in categories that are wide spread over Wikipedia's category graph and cannot be grouped easily. The bootstrapping results of table 7 on page 44 indicated that the former case is more likely.

Organisations are classified correctly with a chance of 67.27% (cosine) and 64.55% (Dice) leaving an error rate of 24.55% (cosine) and 27.27% (Dice). Cosine outperforms the Dice's coefficient on that class.

The CoNLL definitions of *Miscellaneous* do not seem to correspond well with Wikipedia categories. For the evaluation set of type *Miscellaneous* more instances were classified as an organisation in both setups. That indicates a high probability to confuse members of *Miscellaneous* with *Location* which is not that surprising, recalling that the definition of this type is "words of which one part is a location, organisation, miscellaneous or person". Further investigation would be necessary to judge whether type overlaps are just caused by incorrect classifications or if the articles really do belong to that class and maybe should be allowed to be classified as both *Miscellaneous* and *Location*. Remember the example of books dealing with places like "THE RESTAURANT AT THE END OF THE UNIVERSE", which could benefit from a double classification because depending on the context it may serve as the one or the other.

The results of the initialisation step show that the MUC named entity types can be classified with this approach reasonably well with 60.42% (*Location*, cosine) as lower and 81.02% (*Person*, Dice) as an upper bound. This does not work out as well for *Miscellaneous*, but still the lower bound of 35.25% (Dice) beats a baseline with randomly assigned types that would result in 25% correct classifications. Thus, the initially constructed type vectors are useful for NEC of Wikipedia articles. At this time it is not possible to say which of the similarity measures returns better results.

5.3.2 Bootstrapping

The quality of the bootstrapping method is evaluated similar to the initialisation. The members of every annotated set of named entity types were classified by their similarity to the type vectors of every bootstrapping step. The results adapt to the learned vectors and show if the general development of them leads towards the right direction. For every named entity type the following pages present tables with the results for both similarity measures.

Table 9 (page 49) presents the first type *Person*. For cosine similarity we can observe a steady decrease of correct classification from former 78.64% to 73.22% in the last run. *Location* and *Organisation* undergo only small variations after the 4th iteration. From that iteration, no article is left unclassified any more, but the number of articles incorrectly labelled as *Miscellaneous* increases and grows about ten percent. Dice's coefficient seems to be more robust and although its number of correctly classified articles decreases, too, the variation is less than three percent. *Location* and *Miscellaneous* do not change significantly, but we can see that the Dice measure prefers *Organisation* which gains 6% compared to the initial vector. Still, the resulting 78.31% of correct classifications with Dice's coefficient are close to the initial 78.64% using cosine similarity.

Run	***Person***	**Location**	**Organisation**	**Miscellaneous**	**Unclassified**
			Cosine		
initial	232 (78.64%)	17 (5.76%)	25 (8.47%)	19 (6.44%)	2 (0.68%)
1	232 (78.64%)	1 (0.34%)	24 (8.14%)	36 (12.2%)	2 (0.68%)
2	233 (78.98%)	10 (3.39%)	27 (9.15%)	23 (7.8%)	2 (0.68%)
3	216 (73.22%)	9 (3.05%)	23 (7.8%)	46 (15.59%)	1 (0.34%)
4	227 (76.95%)	10 (3.39%)	19 (6.44%)	39 (13.22%)	0 (0.0%)
5	222 (75.25%)	10 (3.39%)	20 (6.78%)	43 (14.58%)	0 (0.0%)
6	222 (75.25%)	10 (3.39%)	21 (7.12%)	42 (14.24%)	0 (0.0%)
7	220 (74.58%)	10 (3.39%)	20 (6.78%)	45 (15.25%)	0 (0.0%)
8	216 (73.22%)	10 (3.39%)	20 (6.78%)	49 (16.61%)	0 (0.0%)
9	216 (73.22%)	10 (3.39%)	19 (6.44%)	50 (16.95%)	0 (0.0%)
10	216 (73.22%)	10 (3.39%)	19 (6.44%)	50 (16.95%)	0 (0.0%)
			Dice's coefficient		
initial	239 (81.02%)	18 (6.1%)	23 (7.8%)	13 (4.41%)	2 (0.68%)
1	239 (81.02%)	14 (4.75%)	37 (12.54%)	3 (1.02%)	2 (0.68%)
2	233 (78.98%)	10 (3.39%)	46 (15.59%)	5 (1.69%)	1 (0.34%)
3	234 (79.32%)	10 (3.39%)	43 (14.58%)	7 (2.37%)	1 (0.34%)
4	235 (79.66%)	14 (4.75%)	40 (13.56%)	6 (2.03%)	0 (0.0%)
5	230 (77.97%)	16 (5.42%)	42 (14.24%)	7 (2.37%)	0 (0.0%)
6	232 (78.64%)	16 (5.42%)	40 (13.56%)	7 (2.37%)	0 (0.0%)
7	232 (78.64%)	15 (5.08%)	41 (13.9%)	7 (2.37%)	0 (0.0%)
8	232 (78.64%)	15 (5.08%)	40 (13.56%)	8 (2.71%)	0 (0.0%)
9	232 (78.64%)	15 (5.08%)	40 (13.56%)	8 (2.71%)	0 (0.0%)
10	231 (78.31%)	15 (5.08%)	40 (13.56%)	9 (3.05%)	0 (0.0%)

Table 9: Similarities created from annotated persons.

Run	**Person**	***Location***	**Organisation**	**Miscellaneous**	**Unclassified**
			Cosine		
initial	0 (0.0%)	116 (60.42%)	21 (10.94%)	14 (7.29%)	41 (21.35%)
1	0 (0.0%)	105 (54.69%)	22 (11.46%)	54 (28.13%)	11 (5.73%)
2	0 (0.0%)	100 (52.08%)	21 (10.94%)	66 (34.38%)	5 (2.6%)
3	0 (0.0%)	101 (52.6%)	18 (9.38%)	69 (35.94%)	4 (2.08%)
4	0 (0.0%)	96 (50.0%)	20 (10.42%)	73 (38.02%)	3 (1.56%)
5	0 (0.0%)	95 (49.48%)	20 (10.42%)	75 (39.06%)	2 (1.04%)
6	0 (0.0%)	93 (48.44%)	19 (9.9%)	78 (40.63%)	2 (1.04%)
7	0 (0.0%)	94 (48.96%)	20 (10.42%)	76 (39.58%)	2 (1.04%)
8	0 (0.0%)	96 (50.0%)	17 (8.85%)	77 (40.1%)	2 (1.04%)
9	0 (0.0%)	97 (50.52%)	18 (9.38%)	75 (39.06%)	2 (1.04%)
10	0 (0.0%)	96 (50.0%)	19 (9.9%)	75 (39.06%)	2 (1.04%)
			Dice's coefficient		
initial	0 (0.0%)	123 (64.06%)	19 (9.9%)	9 (4.69%)	41 (21.35%)
1	0 (0.0%)	147 (76.56%)	26 (13.54%)	7 (3.65%)	12 (6.25%)
2	0 (0.0%)	121 (63.02%)	53 (27.6%)	13 (6.77%)	5 (2.6%)
3	0 (0.0%)	126 (65.63%)	46 (23.96%)	16 (8.33%)	4 (2.08%)
4	0 (0.0%)	123 (64.06%)	47 (24.48%)	19 (9.9%)	3 (1.56%)
5	0 (0.0%)	121 (63.02%)	48 (25.0%)	21 (10.94%)	2 (1.04%)
6	0 (0.0%)	123 (64.06%)	44 (22.92%)	23 (11.98%)	2 (1.04%)
7	0 (0.0%)	124 (64.58%)	39 (20.31%)	27 (14.06%)	2 (1.04%)
8	0 (0.0%)	126 (65.63%)	37 (19.27%)	27 (14.06%)	2 (1.04%)
9	0 (0.0%)	128 (66.67%)	35 (18.23%)	27 (14.06%)	2 (1.04%)
10	0 (0.0%)	128 (66.67%)	35 (18.23%)	27 (14.06%)	2 (1.04%)

Table 10: Similarities created from annotated locations.

Classification results for *Location* can be found in table 10 (page 50). For both kinds of similarity computing, not one run misclassified an instance as a person. With 21.35% this evaluation set has by far the most unclassified articles after the initialisation step. Their count drop down to ∼6% after the first run which is caused by the classification of many locations as described in section 5.2.3 (page 43 ff.). This indicates, that those classifications were correct and increase the quality of the results. Another indicator is that this happens for both similarity values. However, the increase of coverage showed by decreasing counts of unclassified articles in every run is shared by all valuation sets. This was what both similarities have in common. They differ remarkably in the correct classification of *Location*: Cosine's performance is getting worse from the initialisation to the last run with a decline of 10% in correct classifications while Dice's coefficient improves ∼ 2.6% to result in 66.67% in the tenth iteration. Its maximum is 76.56% in the first run. Cosine seems to suffer from its bias towards *Miscellaneous* in this evaluation set. Dice's coefficient shows high variation on classifying the locations incorrectly as *Organisation* in a range from ∼10% to 27.6%.

The next evaluation is done for *Organisation.* Table 12 (page 53) shows the results. Again we can see that the confusion with a person is not likely in both setups. Similar to *Location*, cosine performance decreases with more iterations while Dice's performance raises 10%. In the end, Dice's coefficient outperforms cosine similarity by significant ∼14%. Looking at the other types cosine still favours *Miscellaneous*, while Dice most likely confuses *Location* with *Organisation.*

Run	**Person**	**Location**	***Organisation***	**Miscellaneous**	**Unclassified**
			Cosine		
initial	1 (0.91%)	17 (15.45%)	74 (67.27%)	9 (8.18%)	9 (8.18%)
1	1 (0.91%)	9 (8.18%)	71 (64.55%)	24 (21.82%)	5 (4.55%)
2	2 (1.82%)	11 (10.0%)	66 (60.0%)	29 (26.36%)	2 (1.82%)
3	1 (0.91%)	13 (11.82%)	68 (61.82%)	27 (24.55%)	1 (0.91%)
4	1 (0.91%)	13 (11.82%)	69 (62.73%)	26 (23.64%)	1 (0.91%)
5	1 (0.91%)	14 (12.73%)	67 (60.91%)	27 (24.55%)	1 (0.91%)
6	1 (0.91%)	13 (11.82%)	67 (60.91%)	28 (25.45%)	1 (0.91%)
7	1 (0.91%)	13 (11.82%)	67 (60.91%)	28 (25.45%)	1 (0.91%)
8	1 (0.91%)	14 (12.73%)	66 (60.0%)	28 (25.45%)	1 (0.91%)
9	1 (0.91%)	14 (12.73%)	67 (60.91%)	27 (24.55%)	1 (0.91%)
10	1 (0.91%)	14 (12.73%)	67 (60.91%)	27 (24.55%)	1 (0.91%)
			Dice's coefficient		
initial	2 (1.82%)	21 (19.09%)	71 (64.55%)	7 (6.36%)	9 (8.18%)
1	3 (2.73%)	15 (13.64%)	78 (70.91%)	7 (6.36%)	7 (6.36%)
2	1 (0.91%)	13 (11.82%)	87 (79.09%)	7 (6.36%)	2 (1.82%)
3	2 (1.82%)	20 (18.18%)	84 (76.36%)	3 (2.73%)	1 (0.91%)
4	1 (0.91%)	24 (21.82%)	81 (73.64%)	3 (2.73%)	1 (0.91%)
5	1 (0.91%)	22 (20.0%)	82 (74.55%)	4 (3.64%)	1 (0.91%)
6	1 (0.91%)	21 (19.09%)	83 (75.45%)	4 (3.64%)	1 (0.91%)
7	1 (0.91%)	21 (19.09%)	83 (75.45%)	4 (3.64%)	1 (0.91%)
8	1 (0.91%)	21 (19.09%)	83 (75.45%)	4 (3.64%)	1 (0.91%)
9	1 (0.91%)	22 (20.0%)	82 (74.55%)	4 (3.64%)	1 (0.91%)
10	1 (0.91%)	22 (20.0%)	82 (74.55%)	4 (3.64%)	1 (0.91%)

Table 11: Similarities created from annotated organisations.

Run	**Person**	**Location**	**Organisation**	***Miscellaneous***	**Unclassified**
			Cosine		
initial	1 (0.82%)	10 (8.2%)	47 (38.52%)	46 (37.7%)	18 (14.75%)
1	1 (0.82%)	3 (2.46%)	35 (28.69%)	70 (57.38%)	13 (10.66%)
2	0 (0.0%)	8 (6.56%)	28 (22.95%)	78 (63.93%)	8 (6.56%)
3	0 (0.0%)	9 (7.38%)	29 (23.77%)	78 (63.93%)	6 (4.92%)
4	0 (0.0%)	9 (7.38%)	36 (29.51%)	72 (59.02%)	5 (4.1%)
5	0 (0.0%)	9 (7.38%)	37 (30.33%)	72 (59.02%)	4 (3.28%)
6	0 (0.0%)	9 (7.38%)	36 (29.51%)	75 (61.48%)	2 (1.64%)
7	0 (0.0%)	9 (7.38%)	36 (29.51%)	75 (61.48%)	2 (1.64%)
8	0 (0.0%)	9 (7.38%)	35 (28.69%)	76 (62.3%)	2 (1.64%)
9	0 (0.0%)	9 (7.38%)	36 (29.51%)	75 (61.48%)	2 (1.64%)
10	0 (0.0%)	9 (7.38%)	36 (29.51%)	75 (61.48%)	2 (1.64%)
			Dice's coefficient		
initial	4 (3.28%)	12 (9.84%)	45 (36.89%)	43 (35.25%)	18 (14.75%)
1	4 (3.28%)	11 (9.02%)	44 (36.07%)	48 (39.34%)	15 (12.3%)
2	0 (0.0%)	10 (8.2%)	73 (59.84%)	32 (26.23%)	7 (5.74%)
3	0 (0.0%)	25 (20.49%)	53 (43.44%)	38 (31.15%)	6 (4.92%)
4	0 (0.0%)	28 (22.95%)	50 (40.98%)	39 (31.97%)	5 (4.1%)
5	0 (0.0%)	25 (20.49%)	53 (43.44%)	41 (33.61%)	3 (2.46%)
6	0 (0.0%)	26 (21.31%)	50 (40.98%)	43 (35.25%)	3 (2.46%)
7	0 (0.0%)	22 (18.03%)	53 (43.44%)	45 (36.89%)	2 (1.64%)
8	0 (0.0%)	21 (17.21%)	52 (42.62%)	47 (38.52%)	2 (1.64%)
9	0 (0.0%)	21 (17.21%)	52 (42.62%)	47 (38.52%)	2 (1.64%)
10	0 (0.0%)	21 (17.21%)	50 (40.98%)	49 (40.16%)	2 (1.64%)

Table 12: Similarities created from annotated miscellaneous.

The last evaluation set, *Miscellaneous*, is presented in table 12 (page 53). It has in common with *Location* and *Organisation* that its almost never confused with *Person* independent of the similarity measure. The tendency of cosine similarity to classify more candidates as *Miscellaneous* in every iteration increases its performance in the evaluation to 61.48%. An interesting fact is that the best result of 63.93% is already reached in run two and three. It decreases after that which is remarkable because the majority of articles were added to *Miscellaneous* in iterations four to ten. They do not seem to increase the quality of the category vector. Dice's coefficient improves its performance by ∼5% from the initialisation to the tenth bootstrapping step. Still, it classifies more articles incorrect than correct choosing *Organisation* with a chance of 40.98%.

Finally, the best results after bootstrapping are:

- *Person*: Dice 78.31% (cosine 73.22%)
- *Location*: Dice 66.67% (cosine: 50%)
- *Organisation*: Dice 74.55% (cosine: 60.91%)
- *Miscellaneous*: Cosine 61.48% (Dice: 40.16%)

In conclusion, Dice coefficient performs better than cosine similarity for three out of four named entity types, which implies that taking statistical evidence into account improves the performance of the classification. The numbers indicate that cosine similarity, which ignores the number of occurrences of a category, beats Dice coefficient at the classification of *Miscellaneous* because it is biased.

6 Conclusion and Outlook

In the presented magister thesis we have shown a method to classify more than two million named entities in the multilingual lexical resource HeiNER (Wentland et al., 2008) in two steps, adhering tho the CoNLL definition of named entities (Sang, 2002; Sang and Meulder, 2003). First, we initialised 700,032 classified named entities utilising the category system of Wikipedia starting with a set of 132 manually annotated seed categories. Second, the categories of these classified articles were used to create named entity type vectors to classify yet unlabelled articles by computing the similarities between the vectors and unclassified articles' categories. This was done via bootstrapping in two setups that work with different similarity measures, cosine similarity and Dice's coefficient. The results were evaluated on manually annotated data for both similarity scores. It was shown that the type vectors created from the initialisation step easily outperform a random baseline and that the method is suited well for the named entity types used in MUC-6 (Grishman and Sundheim, 1996) but that the additional CoNLL class *Miscellaneous* shows a gap in quality because it is harder to map the latter to Wikipedia categories. The evaluation of bootstrapping iterations reveals that Dice's coefficient is a better similarity measure than cosine for this particular task. This can be attributed to its property of taking the weights of the vectors' values into account in contrast to cosine's property of only observing the angle between two vectors ignoring their lengths. After all, two lists of named entities were created for each of the types *Person*, *Location*, *Organisation* and *Miscellaneous*, one by cosine and one by Dice similarity.

Additionally we presented mwdb, a new API for Wikipedia written in Python. It provides access to Mediawiki databases in an object oriented way and supports multilingual methods with a comprehensive interface. The aforementioned methods were implemented using mwdb.

Thinking about future work, the quality of the entities could be evaluated externally by applying them in a NERC setup. For example the lists should be useful as a gazetteer feature as it was used by Kozareva (2006) (cf. page 15 ff.). Keeping in mind that HeiNER contains the translations of its named entities, it is easy to generate the gazetteers for other languages as well. By that, maybe it is possible to introduce the gazetteer feature to languages where no comparable resource was available before. The same applies for training data: as HeiNER tracks many contexts per named entity, they can be used as training data for language independent NER systems. This would probably not produce perfect results; still it is an option to get started in a resource poor language with one of the learners presented in section 3.

Also, it would be interesting to investigate the category vectors in detail and show, for example, which categories are the best indicators for which class. The vectors may be used as features as well. E. g. a NER system could look up candidates it has to classify in Wikipedia and compute the similarities to the introduced type vectors. Including the results in the system's feature vector could improve its classification performance.

Because the method to create the initial classified articles only relies on a set of seed categories in Wikipedia it can be applied to fine grained classification of articles with a modicum of effort.

Another interesting option would be to enhance other resources like Wordnet (Fellbaum, 1998), Cyc (Lenat, 1995) or the Wikipedia Taxonomy by Ponzetto and Strube (2007) with the classified instances from HeiNER.

References

E. Alfonseca and S. Manandhar. An unsupervised method for general named entity recognition and automated concept discovery. In *Proceedings of the 1st International Conference on General WordNet, Mysore, India*, 2002.

Oliver Bender, Franz Josef Och, and Hermann Ney. Maximum entropy models for named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*, pages 148–151, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

A.L. Berger, V.J.D. Pietra, and S.A.D. Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):71, 1996.

E. Bick. A named entity recognizer for Danish. In *Proc. of 4th International Conf. on Language Resources and Evaluation*, pages 305–308, 2004.

D.M. Bikel, S. Miller, R. Schwartz, and R. Weischedel. Nymble: a high-performance learning name-finder. In *Proceedings of the fifth conference on Applied natural language processing*, pages 194–201. Association for Computational Linguistics Morristown, NJ, USA, 1997.

Razvan Bunescu and Marius Paşca. Using encyclopedic knowledge for named entity disambiguation. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06), Trento, Italy*, pages 9–16, April 2006.

Davide Buscaldi and Paolo Rosso. Mining knowledge from wikipedia for the question answering task. In European Language Resources Association (ELRA), editor, *Proceedings of the Fifth International Language Resources and Evaluation Conference(LREC'06)*, may 2006.

X. Carreras, L. Marquez, and L. Padro. Named entity extraction using AdaBoost. In *Proceedings of the 6th conference on Natural language learning-Volume 20*, page 4. Association for Computational Linguistics, 2002.

Nancy A. Chinchor. Overview of muc-7/met-2. In *Message Understanding Conference Proceedings MUC-7*, 1999. URL `http://www.itl.nist.gov/iad/894.02/related_projects/muc/proceedings/muc_7_proceedings/overview.html`.

M. Collins and Y. Singer. Unsupervised models for named entity classification. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 189–196, 1999.

Pascal Denis. *New Learning Models for Robust Reference Resolution.* PhD thesis, University of Texas at Austin, 2007.

G. Doddington, A. Mitchell, M. Przybocki, L. Ramshaw, S. Strassel, and R. Weischedel. The Automatic Content Extraction (ACE) Program–Tasks, Data, and Evaluation. *Proceedings of LREC 2004*, pages 837–840, 2004.

Richard Evans. A Framework for Named Entity Recognition in the Open Domain. pages 137–144, 2003.

Christiane Fellbaum. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication).* The MIT Press, May 1998.

J.R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. *Ann Arbor*, 100, 2005.

Michael Fleischman. Automated subcategorization of named entities. In *ACL (Companion Volume)*, pages 25–30, 2001.

Michael Fleischman and Eduard Hovy. Fine grained classification of named entities. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

R. Florian, A. Ittycheriah, H. Jing, and T. Zhang. Named entity recognition through classifier combination. In *Proceedings of CoNLL-2003*, volume 58, 2003.

Ralph Grishman and Beth Sundheim. Message understanding conference: A brief history. In *Proceedings of the 16th International Con-*

ference on Computational Linguistics (COLING), pages 466–471, 1996. http://acl.ldc.upenn.edu/C/C96/C96-1079.pdf.

M.A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics Morristown, NJ, USA, 1992.

Z. Kozareva. Bootstrapping named entity recognition with automatically generated gazetteer lists. In *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 15–21. Association for Computational Linguistics, 2006.

Saul Kripke. Naming and necessity. In Donald Davidson and Gilbert Harman, editors, *Semantics of Natural Language*, pages 253–355. Reidel, Dordrecht, 1972. Reprinted as Kripke (1980).

Saul Kripke. *Naming and Necessity.* Harvard University Press, Cambridge, MA, 1980.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning (ICML)*, 2001.

Seungwoo Lee and Gary Geunbae Lee. Heuristic methods for reducing errors of geographic named entities learned by bootstrapping. In *Second International Joint Conference on Natural Language Processing*, 2005. URL `http://aclweb.org/anthology/I/I05/I05-1058.pdf`.

Douglas B. Lenat. Cyc: a large-scale investment in knowledge infrastructure. *Commun. ACM*, 38(11):33–38, 1995.

Lucian Vlad Lita, Warren A. Hunt, and Eric Nyberg. Resource analysis for question answering. In *The Companion Volume to the Proceedings of 42st Annual Meeting of the Association for Computational Linguistics*, pages 162–165, Barcelona, Spain, July 2004. Association for Computational Linguistics.

C.D. Manning and H. Schütze. *Foundations of statistical natural language processing.* MIT Press, 2002.

J. Mayfield, P. McNamee, and C. Piatko. Named entity recognition using hundreds of thousands of features. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, page 187. Association for Computational Linguistics, 2003.

Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Seventh Conference on Natural Language Learning (CoNLL)*, 2003.

O. Medelyan and C. Legg. Integrating Cyc and Wikipedia: Folksonomy meets rigorously defined common-sense. In *Proceedings of the WIKI-AI: Wikipedia and AI Workshop at the AAAI*, volume 8, 2008.

R. Merchant, M.E. Okurowski, and N. Chinchor. The multilingual entity task (MET) overview. In *Proceedings of a workshop on held at Vienna, Virginia: May 6-8, 1996*, page 447. Association for Computational Linguistics, 1996.

Tom M. Mitchell. *Machine Learning.* McGraw-Hill, New York, 1997.

David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30(1):3–26, January 2007.

Simone Paolo Ponzetto and Michael Strube. Deriving a large scale taxonomy from Wikipedia. In *Proceedings of the 22nd National Conference on the Advancement of Artificial Intelligence (AAAI-07)*, pages 1440–1447, July 2007. URL `http://www.cl.uni-heidelberg.de/~ponzetto/pubs/ponzetto07b.pdf`.

J. Ross Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

Lisa Rau. Extracting company names from text. In *Seventh IEEE Conference on Artificial Intelligence Applications, 1991. Proceedings.*, volume 1, 1991.

M. Ruiz-Casado, E. Alfonseca, and P. Castells. Automatic Assignment of Wikipedia Encyclopedic Entries to WordNet Synsets. In *Advances in web intelligence: Third International Atlantic Web Intelligence Conference, AWIC 2005, Lodz, Poland, June 6-9, 2005: proceedings*, page 380. Springer-Verlag New York Inc, 2005.

Erik F. Tjong Kim Sang. Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *Proceedings of Conference on Natural Language Learning*, 2002.

Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 Shared Task: Language-independent Named Entity Recognition. In *Proceedings of the 7th Conference on Natural language Learning at HLT-NAACL 2003*, pages 142–147, Morristown, NJ, USA, 2003.

Satoshi Sekine and H. Isahara. IREX: IR and IE Evaluation project in Japanese. 2000.

D. Shen, J. Zhang, G. Zhou, J. Su, and C.L. Tan. Effective adaptation of a hidden markov model-based named entity recognizer for biomedical domain. In *Proceedings of the ACL 2003 workshop on Natural language processing in biomedicine-Volume 13*, page 56. Association for Computational Linguistics, 2003.

Y. Shinyama and S. Sekine. Named entity discovery using comparable news articles. In *Proc. the International Conference on Computational Linguistics (COLING)*, pages 848–853, 2004.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 697–706, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-654-7.

G. Szarvas, R. Farkas, A. Kocsor, et al. A multilingual named entity recognition system using boosting and c4. 5 decision tree learning algorithms. *Lecture Notes in Computer Science*, 4265:267, 2006.

Christof Müller Torsten Zesch and Iryna Gurevych. Extracting lexical semantic knowledge from wikipedia and wiktionary. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*. European Language Resources Association (ELRA), may 2008. http://www.lrec-conf.org/proceedings/lrec2008/.

Wolodja Wentland, Johannes Knopp, Carina Silberer, and Matthias Hartung. Building a multilingual lexical resource for named entity disambiguation, translation and transliteration. In European Language Resources Association (ELRA), editor, *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may 2008.

I.H. Witten, Z. Bray, M. Mahoui, and W.J. Teahan. Using language models for generic entity extraction. In *Proceedings of the ICML Workshop on Text Mining*, 1999.

Appendix

A Seed Categories

The seed categories sorted by their named entity type.

A.1 Persons

`People by association`
`People by behaviour`
`People by century`
`People by ethnicity`
`People by gender`
`People by language`
`People by medical or psychological condition`
`People by nationality`
`People by occupation`
`People by place`
`People by political orientation`
`People by religion`
`People by status`

`Human names`

`Fictional characters`

A.2 Locations

`Settlements`
`Regions`
`Fictional locations`
`Astronomical objects`
`Country subdivisions`
`Landmarks`
`Mythological places`

Place names
Paranormal places
Places with restrictions on photography
Protected Areas
Religious places
Rocket Launc sites
Waystations
Buildings and structures

A.3 Organisations

Companies by affiliation
Companies by city
Companies by continent
Companies by country
Companies by industry
Companies by stock exchange
Companies by type
Companies by year of establishment

Corporate subsidiaries by company
Fictional companies
Defunct companies
Private equity portfolio companies
Re-established companies

Royal Warrant

Organizations by activity
Organizations by establishing entity
Organizations by location
Organizations by membership

Collectives
Cooperatives
Foundations

Legal entities

Illegal organizations
Mutual organizations
Non-governmental organizations
Non-profit organizations
Defunct organizations
Proposed organizations

Organizations by subject
Organizations by type
Organizations by year of disestablishment
Organizations by year of establishment
Nobel laureates that are organizations

Communities
Brands

Magazines by country
Magazines by interest
Magazines by language
Magazines by owner
Magazines by publication frequency
Magazines by year of establishment

Defunct magazines
Advertising-free magazines
Alternative magazines
African magazines
Downloadable magazines
Free magazines
Independent magazines
Online magazines
Professional and trade magazines
Propaganda newspapers and magazines

```
Zines

Musical groups by genre
Musical groups by nationality
Musical groups by numbers
Musical groups by time period
Musical groups by year of disestablishment
Musical groups by year of establishment
Musical groups by year of reestablishment

Bands with fictional stage personas
Boy bands
Military bands

Animated musical groups
Child musical groups
Choirs
Family musical groups
Fictional musical groups
Women's musical groups
Musical groups founded by married couples
Supergroups

Musical collectives
Opera companies
Vocal ensembles

Schools
Universities and colleges
```

A.4 Miscellaneous

```
Spiritual theories
Ethical theories
Epistemological theories
Economic ideologies
```

Constructed languages
Ancient languages
Languages by country
Languages by geographical region

Organized events
Events by location
Fictional events
Future events
Current events
Recurring events
Social events
Incidents

Wars by continent
Fictional wars

Sports competitions by sport

Books by author
Books by country
Books by year
Upcoming books

Slogans

Automobile models
Motorcycles by brand
Aircraft by type