Dutch Named Entity Recognition: Optimizing Features, Algorithms, and Output

Toine Bogers
a.m.bogers@uvt.nl

September 8, 2004

Contents

List of Tables

List of Figures

Abstract

Named entity recognition is a subproblem of information extraction and involves processing structured and unstructured documents and identifying expressions that refer to people, places, organization and companies, and so forth. For humans, named entity recognition is intuitively simple. Many named entities are proper names and have initial capital letters and can easily be recognized that way. Memorizing (lists of) names can assist the human recognition process in case of ambiguity.

However, this large amount of ambiguity in natural language makes it difficult to attain human levels of recognition performance. In this thesis we investigate how to optimize the performance of named entity recognition for Dutch texts: what are the best indicators of the named entity type and what approaches maximize the generalization performance? In order to determine the best features for named entity recognition we extract a large number of potentially useful features and performed feature selection experiments using the SFFS algorithm to discover the suboptimal feature sets. To investigate the best approaches to named entity recognition, we select two popular machine learning approaches, memory-based learning and maximum entropy modeling, for our experiments. We also investigate the benefits of splitting the recognition process into separate identification and classification phases as opposed to a one-step process. Finally, we examine the influence of seedlist features and classifier stacking on the generalization performance and attempt to boost this performance by using handcrafted error-correcting rules, essentially creating a hybrid approach with the emphasis on machine learning but with elements from handcrafted approaches.

Morphological features (prefixes and suffixes) turn out to be good indicators of named entity type as well as orthographic features that represent the capitalization characteristics of a word. Seedlist features and stacking features are also very helpful albeit that the actual improvement is highly dependent on the classification algorithm used. The best approach to named entity recognition is a one-step approach using each and every feature in combination with the memory-based learner which outperformed the maximum entropy algorithm in most of the experiments. Nevertheless, splitting the recognition process into separate identification and classification phases appears to be promising as well. Using handcrafted rules to correct frequent errors made by the classifier also turns out to be a successful technique for boosting the generalization performance on the task of named entity recognition.

Chapter 1

Introduction

"Artificial Intelligence is the attempt to get real machines to behave like the ones in the movies."
Russell Beale

A plethora of different definitions of the field of Artificial Intelligence (or AI) exist and most of them are influenced by the experience and personal taste of the author. One of the eminent researchers in the field, John McCarthy, defines AI as "the science and engineering of making intelligent machines, especially intelligent computer programs" [McCarthy, 2003] and Borthwick defines the goal of AI as solving "problems that are easy for humans but hard for computers" [Borthwick, 1999]. A whole range of subtasks is encompassed by these definitions such as pattern recognition, robotics, expert systems, and natural language processing.

The natural language processing (NLP) task is centered around designing and building software that can analyze, understand, and generate natural language. Many of the problems in NLP are part of Borthwick's range of problems that are fairly easy for a human to solve, but devilishly hard for a computer. This difficulty is due to the fact that 'understanding' language means, among other things, knowing what concepts a word or phrase stands for and knowing how to link those concepts together in a meaningful way.

We can distinguish two broad approaches in the NLP community which tackle these problems in a decidedly different manner [Jackson and Moulinier, 2002], although hybrid approaches are not uncommon either. The first approach to NLP is rooted in linguistic analyses and heuristics. It consists for the most part of sets of rules for the manipulation of symbols that say, for instance, whether or not a sentence if well-formed. This rulebased or handcrafted approach imposes these rules on texts top-down. The second approach to NLP is centered around the statistical analysis of language and works bottom-up by looking for patterns and relationships to model using statistical models and machine learning algorithms.

In this thesis we will address one of the subproblems of NLP—Named Entity Recognition or NER—using the latter, statistical approach. Feature selection methods will be used to determine good feature sets for the NER task and two different machine learning algorithms will be used to optimize the classification performance of the task. In the remainder of this chapter an introduction to the NER task will be given and the task will be given a proper place in the NLP hierarchy. The necessary characteristics of the problem will be discussed to introduce the terms necessary to formulate the research question.

The remainder of this thesis is organized as follows. The next chapter describes the extensive body of prior work in the field of NER and explains what approach was used in this thesis and why. Chapter three describes the feature selection problem in greater detail and several solutions to it so that a good feature set can be determined. Chapter four discusses the different machine learning algorithms used in the NER task and the parameter optimization process, and describes the experimental setup and the datasets. In the final two chapters the results are discussed and conclusions are drawn followed by recommendations for future research.

1.1 Information Extraction

Many different subfields exist within Natural Language Processing and one of the subfields of NLP is information extraction (IE). Information extraction deals with finding specific facts from collections of unstructured documents. Exactly what this information is that needs to be extracted is wholly dependent on what constitutes an interesting fact for the users of a system. For instance, intelligence agencies have been using information extraction techniques to examine news feeds and communications traffic since the 1970s. Programs would mark the documents that contained key terms such as "terrorist" or "bomb" after which information analysts would examine these documents [Jackson and Moulinier, 2002].

Modern systems build on this by not only marking relevant documents but actually identifying, extracting and presenting relevant and interesting content. Since information extraction relies for the most part on a statistical approach to NLP there is no real 'understanding' of the text involved; for the most part it is simply a matter of recognizing linguistic patterns. Indeed, research has shown that shallow parsing is sufficient for tasks such as information extraction or question answering [Buchholz, 2002]: complete syntactic analyses are often not necessary for acceptable performace on these kinds of tasks. However, a real understanding of texts is still required for the best possible solutions to these kind of tasks.

Information extraction can be attempted with varying degrees of sophistication and one of the most 'ambitious' forms of information extraction is event extraction. This involves retrieving a wide variety of information about a certain type of event from a document. This task was one of the focal points of the Message Understanding Conferences (MUC) over the years, particularly MUCs 3 through 5 (1991-1993). For instance, the MUC-3 event extraction task concentrated on finding details of terrorist attacks in newswire documents such as date, location, target, instrument and actor. These MUCs and other similar conferences will be discussed in section 1.3.

1.2 Named Entity Recognition

Named Entity Recognition (NER) is a subproblem of information extraction and is, as such, slightly less complex than IE. Indeed, for the location, target or actor of a (terrorist) event to be extracted, these (named) entities first need to be recognized as such. This NER task (also known as 'proper name classification') involves the identification and classification of so-called named entities: expressions that refer to people, places, organizations, products, companies, and even dates, times, or monetary amounts.

This in turn means that every word needs to be categorized as belonging to a named entity or not. In other words, not only the boundaries of these named entities should be determined but also the type of named entity. Named entities can consist of any type of word: adverbs, prepositions, adjectives, and even some verbs, but the majority of named entities are made up of nouns. Named entities themselves are also usually noun phrases, but they can also occur as certain adjectives such as "Chinese" [Borthwick, 1999, Jackson and Moulinier, 2002].

Named entity recognition is intuitively simple for humans and indeed building a NER system with reasonable performance is fairly easy. Many named entities are proper names and have initial capital letters and can easily be recognized that way. Performance can be further improved by using so-called *gazetteers* or *seedlists*: lists of people, places and companies. A sentence such as (1) can be tagged quite easily using these simple rules and lists. Note that from this point on, every named entity in the examples will be marked by a grey box.

(1) Canada will spend $38-million on sending troops to Haiti in March 2004, says Jean Chretien.

However, attaining human performance levels of NER is still difficult due to the large amount of ambiguity in natural language. A name such as "Washington" could refer to either a person, a city, or a state while the words "New York" in the franchise name "New York Pizza" should not be marked as a city but as part of the name of the organization.

Furthermore, gazetteers are not always helpful if the same word occurs in more than one of these lists. Sentence (2) is an example of more ambiguous input for the NE recognizer:

(2) Washington refused to comment on the anticipated takeover of Petaluma Accounting by Arthur Andersen next fall.

This sentence is problematic for several reasons: as mentioned before, the name "Washington" can refer to a person or a location; even in sentence (2) marking it as a person is not one hundred percent certain. Furthermore, the word is at the beginning of the sentence, so capitalization information is useless here. The word "Petaluma" should not be tagged as a location but as part of the company name "Petaluma Accounting" and "Arthur Andersen" is not a person but a company in this case. Finally, the word "next" is part of the temporal expression "next fall" and should be marked as such.

Many different approaches exist for dealing with these problems and ambiguities and the divide between the handcrafted and the machine learning approach to NLP is also visible in these different approaches to NER. An overview of the prior work in the field of NER will be given in the next chapter.

1.2.1 Applications of Named Entity Recognition

As mentioned earlier, shallow parsing is held to be sufficient for many NLP tasks and this is exactly what NER does in the sense that it delimits sequences of tokens that represent the who, where, what and how elements in a text. This makes NER is an important pre-processing task for many other information extraction tasks such as machine translation, information retrieval, and question answering. In the latter task, for instance, the different named entities could be linked together to gain a better understanding of the questions and of text in general [Bikel et al., 1997].

Another possible application of NER technology would be to use it to enhance and enrich the use of information retrieval systems such as (Internet) search engines in several ways. Search engines themselves could use NER to rewrite queries like "Where does Wayne Gretzky live?" into the more appropriately-formed query `"Wayne Gretzky"+live`. Users themselves can also better utilize search engines by explicitly specifying that the query "Clinton" means that they are looking for more information about Clinton, South Carolina and not former president Bill Clinton, for instance.

Automatic indexing of (web) documents can also be improved since for many documents the majority of index terms are named entities. For instance, this would enable users to call up all documents on a corporate intranet pertaining a certain individual. Magazines like "The Sporting News Hockey" or "National Geographic" could even use NER to automatically mark up names of athletes or geographic locations in a different font. NER-aided markup could even be utilized online to link directly to specialized pages about these named entities [Bikel et al., 1997, Borthwick, 1999].

1.3 Evaluation

Recent progress in NER and information extraction tasks in general has been greatly facilitated by the annual workshops and conferences about these topics such as the Message Understanding Conferences (MUC), the Conferences on Natural Language Learning (CoNLL) and the Japanese Multilingual Entity Tasks (MET). These workshops and conferences are meant to support research within the IE community and they have made a successful effort to encourage experimentation with test collections of realistic size. And while each conference has its own set of subgoals, what all of these conferences have in common, is that they organize so-called shared tasks that rely on an agreed set of metrics and the availability of a uniform set of training and test data to encourage rigorous evaluation and a thorough evaluation process. Many advances in the field of NER have been presented on these conferences.

The MUC conferences consisted of a series of seven annual conferences and were initiated in 1987 by the Defense Advanced Research Projects Agency (DARPA) as a means of evaluating practical running information extraction systems in a carefully controlled test environment followed by a conference in which the participants presented papers discussing their methods. The MUCs were modeled after the Text REtrieval Conferences (TREC), another initiative started by the US government in 1992 to support research in the information retrieval community by providing the infrastructure necessary for large-scale evaluation of text retrieval methodologies.

The last two MUC-6 and MUC-7 conferences were very beneficial to the scientific progress made in the field of (English) NER. In the shared tasks[1] of these two MUCs not only named entities (person, location, and organization) had to be recognized, but also temporal expressions (date, time, and duration) and numeric expressions (monetary or cardinal amounts, measurement phrases, and percentages). The MET conferences mimicked the format of the MUC conferences and had the same positive impact on Japanese NER.

The CoNLL conferences are an initiative of the ACL's Special Interest Group on Natural Language Learning—or SIGNLL—and have been an annual event since 1997. The 2002 and 2003 shared tasks dealt with language-independent NER and the usefulness of additional unannotated data and some of the other shared task topics in recent years were clause identification, chunking, and semantic role labeling. The NER experiments described in this thesis were performed on the Dutch dataset provided for the 2002 CoNLL shared task and therefore the next subsection is devoted to a more extensive description of this specific shared task.

1.3.1 CoNLL-2002

The shared task of CoNLL-2002 concerned language-independent named entity recognition and concentrated on four types of named entities: persons, locations, organizations and names of miscellaneous entities that did not belong to the previous three groups. Participants were required to use the annotated data for developing a named entity recognition system that included a machine learning component. One of the points of interest was how additional unannotated data could be used for improving their performance [Tjong Kim Sang, 2002a].

The whole shared task process consisted of several different stages. First, annotated training and test datasets were provided for Dutch, the language of interest in this thesis, and for Spanish. More information about the Dutch datasets can be found in chapter 4. The datasets were split into a training set, a development set, and a test set and distributed to all participants who proceeded by training their learning methods on the training set. The development data was available to help participants tune the parameters of their learning methods. When the best parameters had been found, the resulting NER system would be trained on the training data and tested on the test data. The results of the different approaches were then submitted and compared in the evaluation of the shared task. The split between test and development data was performed to avoid overfitting; this way the systems would not be tuned to the test data [Tjong Kim Sang, 2002a].

[1] Shared tasks are scientific competitions organized by the conferences where participants can test their ideas about approaches to specific IE problems.

In the evaluation of the results, the named entities were considered to be recognized correctly if they were an exact match of the corresponding entities in the data file. In contrast, named entities in the MUC-6 and MUC-7 NER tasks were scored on both tag and type; a correct tag meant they had the correct start and end points, and a correct type meant that the type of entity was identified correctly [Borthwick, 1999]. Two well-known performance metrics that are often used in evaluating such classification results are precision and recall. Precision is defined as the ratio or percentage of named entities found by the system that is correct. Recall is the ratio or percentage of named entities present in the corpus that were actually found by the system [Tjong Kim Sang, 2002a].

	Correct	**Incorrect**	
Retrieved	a	c	$a + b = m$
Not retrieved	b	d	$c + d = N - m$
	$a + c = n$	$b + d = N - n$	$a + b + c + d = N$

Table 1.1: A contingency table analysis of precision and recall for the NER task.

For another explanation of the concepts of precision and recall, observe the contingency table of possible outcomes in table 1.1 [Jackson and Moulinier, 2002]. When regarding this table, precision is given by $P = \frac{a}{n}$ and recall by $R = \frac{a}{m}$. Many different composite measures exist that relate these measures of precision and recall and one of the more accepted measures is the F_β-measure of [Van Rijsbergen, 1975]. This F_β-measure represents the weighted harmonic mean of precision and recall and is given by:

$$F_\beta = \frac{\left(\beta^2 + 1\right) \cdot P \cdot R}{\beta^2 \cdot P + R} \tag{1.1}$$

with $0 \leq \beta < \infty$. The β is used to specify the relative importance of recall over precision. With a β of 0 the F_β-measure is equal to the precision and larger values of β attach a greater importance to recall [Rennie, 2004]. The standard value for β is 1 so that equal weight is given to both precision and recall:

$$F_\beta{}' = \frac{2 \cdot P \cdot R}{P + R} \tag{1.2}$$

This simplified formula was used to score the different systems of the participants of the CoNLL-2002 shared task and it was also used in the evaluation of, for instance, MUC-7. Consequently, this is also the measure we will use to evaluate the experiments described in this thesis since they were performed on the Dutch CoNLL-2002 dataset.

Of all participants in the 2002 CoNLL shared task, the team of Carreras, Màrquez, and Padró obtained the best results on the Dutch NER task with a score of $F_{\beta=1} = 77.05$ on the test set [Carreras et al., 2002, Tjong Kim Sang, 2002a]. At first glance this score seems average compared to the highest scores obtained on the English MUC tasks, which range from 85% to 93%. However, English NER has received a great deal more attention than NER in other languages and the goal of the MUC tasks was never to develop language-independent systems but to achieve the best

possible performance on English documents.

It should also be noted that trying to achieve a perfect score on any named entity recognition task is an unrealistic goal, since human performance on the NER task is not perfect either; according to Palmer human scores range from 96% to 98%. The fact that human performance is never perfect is a common point of reserve for all information extraction tasks [Palmer and Day, 1997].

1.4 Research questions

The deliberate focus on language-independence of the systems participating in the 2002 and 2003 CoNLL shared NER tasks automatically entails a lack of fine-tuning on a specific language such as Dutch. Many different learning methods were employed and a wide variety of information were extracted from the dataset in order to classify the named entities correctly, but no conclusive overview of the best features was given.

In this thesis we will try to determine exactly what kind of (word) information has the best predictive value for classifying Dutch named entities. We will use two algorithms that have been known to perform well on IE extraction tasks such as maximum entropy and memory-based learners. Which combination of learning method and its parameters yields the best results will be also be examined. Recent developments have shown that this is far from easy due to the complex interdepencies between feature selection and parameter optimization. In short, one of the research questions we pose in this thesis is:

> Given two machine learning algorithms and a pool of potentially useful features, what are the best indicators of the named entity type, and what algorithms and algorithm parameters should be used to maximize the generalization performance of Dutch NER?

Another open issue regarding the task of NER is whether or not recognition should be immediate or split into a identification and classification stage. Both approaches occur frequently in other NER work but no satisfying answers have been given to this question. Therefore, this is the second research question we attempt to answer in this thesis:

> Should the named entity recognition process be split into separate identification and classification phases or should these two tasks be combined and take place in one pass?

Chapter 2

Approaches to Named Entity Recognition

Many different approaches to named entity recognition have been undertaken in the past decade and a large number of different systems have been proposed and submitted to the shared tasks of the CoNLL and MUC conferences. In this chapter, we will attempt to fit the most promising of these systems and approaches in a hierarchy that reflects the differences between handcrafted and machine learning approaches that is visible in NLP in general. Finally, we will also discuss the merits of separating NE identification and classification vs one-shot classification.

2.1 Handcrafted approach

Many of the systems that participated in the MUC shared tasks for NER used a large number of handcrafted rules that perform simple pattern matching. This in contrast to the systems participating in the CoNLL that were required to contain a machine learning component; systems using only sets of handcrafted pattern-matching rules were not allowed to participate.

Handcrafted systems rely for a great deal on the human intuition of their designers who construct a large number of rules that capture the intuitive notions that come to mind when contemplating a simple approach for recognizing named entities. For instance, in many languages it is quite common for person names to be preceded by some kind of title. Many variations on examples like (3b) could therefore easily be covered by rules like (3a) [Grishman, 1995].

(3) a. `title capitalized_word` ⇒ `title PERSON`

b. The man introduced himself as Dr. Venkman .

Another example of when the tell-tale left or right context of an expression can help to classify a named entity is the situation where locations are preceded by location-related nouns such

as "district" or "region" like in example (4b). A rule that could resolve some of the ambiguity present in example (2), repeated here in part as (5b), could check for patterns related to corporate takeovers.

(4) a. `location_marker "of" capitalized_word` ⇒
`location_marker "of" LOCATION`

b. He grew up in the district of Bodega Bay .

(5) a. `"takeover of" capitalized_word "by" capitalized_word` ⇒
`"takeover of" COMPANY "by" COMPANY`

b. [...] the anticipated takeover of Petaluma Accounting by Arthur Andersen [...]

As mentioned before, many of the systems that participated in the MUC shared tasks used this approach such as the NE system of Black, Rinaldi, and Mowatt [Black et al., 1998]. Their system was actually a module of a larger system called FACILE whose main purpose was filtering of news by fine-grained knowledge-based categorization. The FACILE NE module did not employ any learning methods, but was completely rule-based and explicit weights were assigned to the rules to enable the module to choose between competing analyses. The module also included a method for identifying name-strings as coreferential with longer variants in the same text. Below is an example rule from this module that looks for university names and classifies them as organizations: if a known country, region, or city is preceded by the expression "university of", that whole expression should be marked as an organization with a confidence factor of 0.9.

```
[syn=NP, sem=ORG] (0.9) =>
 \ [norm="university"],
   [token="of"],
   [sem=REGION|COUNTRY|CITY] /;
```

Other systems that made extensive use of handcrafted rules in the MUC shared tasks were NYU's Proteus system [Grishman, 1995] and the IsoQuest's NetOwl extractor system which was also in commercial use in 1998 [Krupka and Hausman, 1998]. In all of these systems each rule was handcrafted although it is certainly possible to automatically extract these patterns from training material. One advantage of the handcrafted approach is that knowledge is clearly represented, stored in, and retrievable from the systems.

However, a big disadvantage of the handcrafted approach is that certain rules may work well for certain text domains but it is questionable whether most rules would carry forward into a new domain. This means that the entire knowledge base of a handcrafted rulebased system has to be rewritten if the system is to be applied to a new domain. In addition to the lack of portability, constructing any rule base of effective size is very expensive in terms of time and money: both the NYU and the IsoQuest reported spending a person-month each just on writing the rule base. Furthermore, this enormous expense is required every time the system needs to be ported to a new domain [Borthwick, 1999].

2.1.1 Sequence strategy

Despite the obvious disadvantages of the handcrafted approach, the best MUC-7 system employed a variety of handcrafted rules and heuristics in a sophisticated way. This system came from Edinburgh University and was developed by Mikheev, Moens, and Grover. Even though their system includes some probabilistic techniques, it warrants a separate mention as a handcrafted approach.

The system made use of *internal* (phrasal) and *external* (contextual) evidence and this works as follows: certain word strings have a structure that suggests that they are named entities such as "Arthur Andersen" in (5b), but not the type of named entity. Seedlists may not always be helpful here in this, since "Arthur Andersen" may occur in multiple seedlists. However, somewhere in the text there is likely to be some piece of contextual information which makes it clear what type of named entity it is. The idea is to delay the final classification of a named entity until that piece of contextual information has been found. The approach was dubbed the 'Sequence Strategy' and consists of five different steps [Mikheev et al., 1999]:

- **Sure-fire rules**
  The first step of the Sequence Strategy involves the application of sure-fire rules. Only if a word or string of words occurs in an unambiguous context and if there is no contradictory evidence such as presence in more than one seedlist, will a named entity be marked up. At this stage, every assignment is considered to be likely and not definite.

- **First partial matching**
  In the second stage the system collects all the named entities that have already been identified and generates all possible partial orders of the composing words preserving their orders and marks them as possible matches if they are found anywhere else in the text. So if the string "Arthur Andersen Consulting" was marked as an organization in the first stage, all occurrences of for instance "Arthur Andersen" and "Andersen Consulting" are also tagged as possible organization.
  All this information goes into a pre-trained maximum entropy model which takes into account different contextual information such as position in a sentence and whether they occur in lowercase somewhere else in the document. If the model positively classifies a partial match, the system makes a definite assignment.

- **Rule relaxation**
  In this stage the system once again applies the grammar rules of the first stage but the contextual constraints are relaxed. Seedlists are also used without reserve to mark up named entities including person names, for ambiguous named entities that look like organizations or locations will have been filtered out in the first two steps. Conjunctions will also be resolved at this stage; for instance, for "Dunst and Duskhu Accounting" the system will check if the possible parts of the conjunction have been used as separate named entities elsewhere in the document.

- **Second partial matching**
  At this point all resources such as internal evidence and seedlists have been exhausted. In the fourth stage the system once again generates all possible partial orders of the named entities already identified and marks them as named entities of the same type after which the pre-trained maximum entropy model is used again in the assignment of the tags.

- **Title assignment**
  Document titles often have different capitalization rules regarding named entities and in the final stage all named entities in the title are identified using the results of the classification of named entities in the body of the document.

2.2 Machine learning approach

As mentioned before in the previous chapter, the machine learning approach to the analysis of language works bottom-up by looking for patterns and relationships to model. A great many different learning methods for modeling these relationships exist and the most popular and most significant machine learning approaches will be discussed here.

A common distinction between these different learning methods is that of supervised and unsupervised learning methods and the methods will be given a place in this hierarchy. *Supervised learning* involves using a program that can learn to classify a given set of labeled examples that are made up of the same number of features. Each example is thus represented with respect to the different feature spaces. The learning process is called supervised because the people who marked up the training examples are teaching the program the right distinctions. This is different from *unsupervised learning* where a program learns without feedback, for instance, by clustering similar documents or entities together [Jackson and Moulinier, 2002].

2.2.1 Supervised learning

Supervised learning methods can be further categorized with respect to the moment generalization beyond the training data takes place. So-called *eager learners* commit their generalization model to memory after training before new query instances are encountered, while *lazy learners* defer this generalization until new query instances are encountered. This deferment of generalization implies that lazy learning methods will generally require less computation during training and more computation when presented with a new query for which they have to predict the output class. The situation is reversed for eager learning methods [Mitchell, 1997].

A popular supervised learning approach that warrants a separate mention is that of combining different classifiers. This approach is quite popular in NER and it has been argued that it presents a definite improvement over single-classifier approaches [Hendrickx and Van den Bosch, 2003, Carreras et al., 2002, Florian, 2002, Wu et al., 2002].

Eager learning methods

A specific group of eager learning methods that are a popular choice for NER are pure probabilistic models such as, for instance, Hidden Markov Models (HMM). In the past couple of decades, HMMs and other probabilistic modeling methods have enjoyed great success in other NLP problems such as speech recognition and part-of-speech tagging [Church and Mercer, 1993]. The underlying assumption of the HMM approach to Named Entity Recognition is that a text originally had all the named entities marked for convenience before it was passed through a noisy channel where the markup information was somehow lost. The goal is then to model the original process that marked up the names [Jackson and Moulinier, 2002].

One of the best performing NER systems at MUC-6 and MUC-7 was a system developed by Bikel et al. called Nymble. Other HMM approaches to NER are those of Jansche, Bender, and Malouf for their entries in the 2002 and 2003 CoNLL shared NER tasks. In a standard HMM approach the tag probabilities depend on the current word and the previous tag. All systems used a modified version of these HMMs that employed a variety of different word features leveraged with information about word position and adjacency to estimate the different named entity classes [Bikel et al., 1997, Bender et al., 2003, Jansche, 2002, Malouf, 2002b].

A disadvantage of an HMM is the independence assumption: word/tag probabilities and the tag sequence probabilities are assumed to be independent; that is, a word's most likely named entity type is assumed to be independent of the surrounding named entities. This is clearly a big obstacle. *Maximum entropy models* can handle a large number of features which can be used to overcome some of the problems of HMMs. Maximum entropy models are a class of exponential models which require no unwarranted independence assumptions. They have proven to be very successful in general for integrating information from disparate and possibly overlapping sources. Maximum entropy modeling is one of the algorithms used in the experiments described in this thesis and therefore it will be discussed in greater detail in chapter 4.

Malouf also used a maximum entropy model in his 2002 approach and in general many systems use maximum entropy models, not only for NER, but for a whole range of NLP problems. It was the most frequently applied technique in, for instance, the CoNLL-2003 shared task where five systems used this statistical machine learning method. Maximum entropy models seem to be a good choice for the NER task since the top three results for English and the top two results for German were obtained by participants who employed them in one way or another.

NYU's entry in the MUC-7 shared task was called MENE and used maximum entropy models as well to great effect, demonstrating that these learning methods provide a high degree of transdomain and translingual portability. It also showed a solid performance on the recognition task not only on the MUC-7 task but also on the two Japanese named entity tasks it was used for [Borthwick et al., 1998].

Machine learning also has a branch of supervised algorithms which encompass a large number of different eager learning methods. Some of the more popular approaches will be discussed here. Decision tree learning is one of the more popular supervised learning methods and it in-

volves classifying instances by constructing a tree that incorporates just those feature test needed to discriminate between the instances and their classes in the training data. Each node in the tree specifies a test of some instance attribute and all branchings from that node correspond to one of the possible values of that attribute. Classifying new instances is then simply a matter of traveling down the tree through each feature test until an output class is assigned in the leaf node at the end of the sequence of tests.

Paliouras et al. examined the use of the decision tree learner C4.5 in porting NER systems from one specific domain to the next and they showed that the resulting decision trees can outperform manually constructed sets of rules [Paliouras et al., 2000]. In their 2002 CoNLL shared task submission, Black and Vasilakopoulos compared decision tree learning with transformation-based learning and found that, although the decision tree learner performed adequately, the transformation-based learner outperformed it on the NER task.

Transformation-based learning (TBL) is an error-drive machine learning technique that is somewhat similar to decision tree induction in that the end result of the latter can be seen as a set of rules that govern the classification process; deriving such a set of transformation rules is the explicit goal of TBL. First, an initial classification is assigned to the data after which the best possible transformation rules are proposed, evaluated and selected that maximally decrease the number of errors [Brill, 1995]. Aside from Black and Vasilakopoulos, Florian also used TBL as one of the classifiers in his combined approach [Florian, 2002].

A large number of other machine learning approaches have been applied to the problem of NER such as support vector machines [McNamee and Mayfield, 2002, Hearst et al., 1998] and various connectionist approaches [Florian, 2002, Hammerton, 2003] but these are not as prolific as the other approaches so they will not be discussed in greater detail here.

Lazy learning methods

Several different lazy learning methods exist but only memory-based learners have been used regularly for NER and other NLP tasks with favorable performance compared to other learning methods. A memory-based or instance-based learner stores the training examples and their classes and compares new query instances to each of the stored examples by computing the similarity (or distance) between each of the stored examples and the query instance. It then assign the new query instance the class of the most similar example(s). Several different methods for choosing from the similar examples and for computing their similarity with the query instance and the performance of the memory-based learner crucially depends on the choice of metrics and selection methods.

Many different participants used memory-based learners in one way or another in their NER systems; [Hendrickx and Van den Bosch, 2003], [De Meulder and Daelemans, 2003], and
[Tjong Kim Sang, 2002b] all used memory-based learners in the systems they submitted to the CoNLL shared tasks.

Combining classifiers

As mentioned before, a popular approach that warrants a separate mention is that of combining different classifiers, also referred to as *meta-learning*. The concept of *stacking* involves combining the predictions from multiple classifier models and is particularly useful when the types of models are very different. Several different stacking methods exist and arguably the simplest way of combining classifiers is simply stacking one on top of the other, also known as second-stage stacking. For instance, the results of one classifier can be used as extra features for another classifier, and the predictions of a classifier can even be fed back to the classifier itself. Stacking is quite popular in the CoNLL shared NER tasks: Hendrickx, Florian, Mayfield, and Klein all used stacking in their systems [Hendrickx and Van den Bosch, 2003, Florian, 2002, Mayfield et al., 2003, Klein et al., 2003].

Another approach that combines classifiers is *bagging* which is a classic voting scheme that usually works best with so-called unstable classifiers—weak classifiers with a high variance. The training material is resampled several times using bootstrapping and the weak classifier is trained on each resample. Some weighted combination of the different predictions is then used to determine the final classification [Hsu, 2001, StatSoft, 2003]. Bagging seems to be a less popular approach for NER; nevertheless, Munro et al. used bagging in their 2003 submission for the CoNLL shared task [Munro et al., 2003].

A more sophisticated algorithm for generating weights for weighted prediction or voting is the *boosting* procedure. Boosting has a novel feature in that it associates weights with training instances. The other meta-learning approaches discussed so far all treat each training instance in the same way. A weak learner is trained on the training data for several boosting rounds and a set of weights over the training instances and labels is maintained. As training progresses, training examples and corresponding labels that are hard to predict get higher weights while the 'easier' examples get lower weights. This means that each round the weak learner concentrates on those documents that are hardest to classify; easy parts of the problems are solved by a simple classifier, and harder parts are solved with more effort [Jackson and Moulinier, 2002].

This approach has turned out to be very effective for many classification tasks including NER; boosting is a very popular approach in the CoNLL shared tasks; Wu, Carreras and Collins all used boosting in one way or another to combine their classifiers [Wu et al., 2002, Wu et al., 2003, Carreras et al., 2002, Carreras et al., 2003, Collins, 2002].

2.2.2 Unsupervised learning

In unsupervised learning the goal of the program is to build representations from data, for instance, by clustering similar documents or entities together or reducing the dimensionality of the data. These representations can then be used for data compression, classifying, decision making, and other purposes. Unsupervised learning is not a very popular approach for NER and the systems that do use unsupervised learning are usually not completely unsupervised. For instance,

Cucchiarelli and Velardi employed unsupervised learning techniques to acquire contextual classification cues and then use the results of this phase to classify unrecognized proper nouns in an unlabeled corpus, thereby complementing an existing supervised NE recognizer. The contextual model of proper noun categories is learned without supervision [Cucchiarelli and Velardi, 2001].

However, some researchers have used unsupervised learning as their main approach of classifying named entities with success. Cucerzan and Yarowsky used a minimally supervised learning algorithm—also known as *bootstrapping*—that used short name lists as seed data and morphological and contextual evidence to create hierarchically smoothed trie structures to model the structure of different types of named entities [Cucerzan and Yarowsky, 1999]. Collins and Singer also used morphological and contextual evidence in their unsupervised learning approach to NER [Collins and Singer, 1999]. Finally, the hybrid approach of [Buchholz and Van den Bosch, 2000] combines unsupervised and supervised techniques.

2.3 Separating named entity identification and classification

An interesting problem in NER is whether or not the process should be split into separate identification and classification stage or whether these two steps should be combined into one complete process. Both approaches have their advantages and disadvantages, and advocates and opponents; however, no conclusive evidence has been presented yet arguing that one of the two approaches is to be preferred over the other.

Arguably the most intuitive way of recognizing named entities would be to employ separate identification and classification steps. In the identification phase the boundaries of the named entities are determined by chunking together the words that make up a named entity. The classification step then consists of assigning NE class labels to all the candidate named entity phrases that were chunked together in the previous step.

The main reason and underlying rationale for separating the recognition process into two distinct identification and classification steps is that the optimal feature sets for both stages are somewhat disparate; determining the boundaries of named entities requires different information than determining its class. By separating the two steps, the chunking program can concentrate on any morpho-syntactic regularities across different sorts of phrases without having to determine the class label yet. In light of this it makes sense to delay the assignment of class labels.

The candidate phrases proposed by the identification component are subsequently classified and the advantage here is that this way we can take a lot more context into account for classifying phrases. It would be cumbersome to include all of the features of both steps at the same time. Some of the proponents of separation are [Carreras et al., 2003], [Jansche, 2002] and [Munro et al., 2003] who all separated the steps in their CoNLL approaches.

However, separating the steps is not without its disadvantages. For one, the separation requires more work and effort than a one-shot approach and this extra effort has to be expended each time the system is ported to a new languages since the same features are likely to have dif-

ferent relative levels of significance in different languages. Furthermore, the recombination of the different models has been shown to propagate errors [Carreras et al., 2002].

For these reasons, several other researchers have opted for the one shot approach where the chunking and labeling steps are combined; among these were [Borthwick, 1999], [Malouf, 2002b], and [Hendrickx and Van den Bosch, 2003]. Both approaches occur frequently in other NER work but no satisfying answers have been given to this question which of them is to be preferred. Therefore, this is one of the research questions we attempt to answer in this thesis.

Chapter 3

Feature selection

Several steps have to take place before the named entity recognition process can proceed successfully, irrespective of whether it is split up into two phases or combined in a single phase. First, example data needs to be collected, cleaned and prepared before it can be split into training and test data. The first step that the participants themselves have to take is transforming the data into a format appropriate for their system. This is necessary for both handcrafted as well as machine learning approaches and does not only involve cleaning up the data but also extracting the desired features from the data that are necessary for the recognition process. In many real-world situations, including NLP problems, relevant features are often unknown a priori. Therefore, a large number of mostly weakly predictive, candidate features are introduced to better represent the domain. The problem of determining which features have the greatest predictive value is called *feature selection* and is defined as follows: given a set of candidate features, select a subset that performs the best on unseen examples using some classification system. Performing feature selection can have several advantages:

- It reduces the cost of recognition and the execution time by reducing the number of features that need to be collected.
- In some cases feature selection can also provide a better generalization performance due to the exclusion of irrelevant or redundant variables.
- Knowing which features are relevant can give insight to the nature of the problem at hand.

Feature selection can be performed manually but this relies heavily on the intuition of the human feature selector and manual feature selection is simply unpractical for many problems. Many different automatic feature selection methods have been proposed in the past couple of decades and we will discuss some of the most important ones in this chapter.

The final section of this chapter will focus on the complex interdepencies between feature selection and parameter optimization. Different learning methods and even different parameter values of one learning method can have different optimal feature sets. This complicates the search for the optimal feature set. We discuss several ways to counteract these complications and we describe our method for solving this problem.

3.1 Feature selection methods

As mentioned before, feature selection involves searching through subsets of features and trying to find the best one by weeding out or avoiding the many partially or completely irrelevant or redundant features. An irrelevant feature does not affect the target concept in any way, and a redundant feature does not add anything new to the target concept.

Generating and examining each and every subset in the search process would be too costly and practically prohibitive, even for a medium-sized feature set size (N), as it would require evaluating 2^N subsets. Therefore, most feature selection methods examine only a smaller part of the feature subset space, which in turn means that the resulting subset need not be optimal. Most feature selection methods search for a suboptimal subset of features in that the selected subset may be the optimal subset for that feature selection method eventhough another selection method might yield a different suboptimal subset. According to [Dash and Liu, 1997], there are four basic steps to a typical feature selection method:

1. A *generation procedure* to generate the next candidate subset; this generation procedure is a search procedure that generates feature subsets for evaluation. Since generating each and every subset was deemed to be impractical, this means that there are basically three other possibilities: the procedure can start with no features, all features or with a selected subset of features. In the first two cases, features are iteratively added or removed; in the last case, features can be added or removed iteratively or randomly.

2. The *evaluation function* measures the value of the subset under examination produced by some generation procedure. This value is compared with the previous best subset and if it is found to be better, then the previous best subset is replaced by the new set. The final, optimal subset is always relative to a certain evaluation function and typically, an evaluation function tries to measure the discriminating ability of a feature or a subset to distinguish the different class labels.
   Evaluation functions can be divided into two broad categories based on whether or not they incorporate the inductive algorithm that will finally use the selected subset. In the so-called *wrapper* approach, the learning method that one ultimately intends to use is utilized already in the feature selection phase by using it as the evaluation function itself [Reunanen, 2003].
   Filter methods are the other category of evaluation functions and these are independent of the inductive algorithm. Filter methods are data intrinsic measures: they focus only on the data itself. For instance, distance measures are used to determine the distance between two features. If the difference is zero, then the features in question are indistinguishable

and redundant. Information measures typically determine the information gain from a feature, which is defined as the difference between the prior uncertainty and expected posterior uncertainty using that feature. The feature that best reduces the uncertainty is the best feature. Correlation measures are somewhat similar to distance measures and qualify the ability to predict the value of one variable from the value of another. Finally, consistency measures are characteristically different from other measures in that they prefer consistent hypotheses definable over as few features as possible. These measures find out the minimally sized subset that satisfies the acceptable inconsistency rate, which is usually set by the user [Dash and Liu, 1997].
The wrapper approach was used for all the feature selection experiments in this thesis because the inductive algorithm is likely to apply some form of filtering itself in the learning process and because the wrapper approach is likely to be closer to the end result since the same inductive algorithm is used there.

3. A suitable *stopping criterion* is necessary to prevent the feature selection process from running indefinitely through the subset space. The choice for a stopping criterion is influenced by the generation procedure and by the evaluation function. If the stopping criterion is based on the generation procedure, then the feature selection process can be halted when a predefined number of features has been selected, or when a predefined number of iterations has been reached. Stopping criteria based on the evaluation function can prompt the feature selection process to cease when addition (or deletion) of any feature does not produce a better subset; or when an optimal subset according to the evaluation function is obtained.

4. Even though the *validation procedure* is not a part of the feature selection process itself, a feature selection method has to be validated by carrying out different tests, and comparing the results with previously established results.

3.2 A taxonomy of feature selection methods

A taxonomy of popular feature selection algorithms into broad categories is presented below in figure 3.1. This taxonomy is by no means complete but complete enough for the purposes of this thesis. We first divide the feature selection methods into two groups of methods: those guaranteed to find the optimal solution and those that may result in a suboptimal feature set.

3.2.1 Optimal

The optimal methods can be further divided into exhaustive methods—which need to examine every single feature subset in order to determine the optimal set—and non-exhaustive methods—that only need to examine a subset of the space of feature subsets. Obviously, exhausting the feature subset space by generating each and every subset is implausible for all but the smallest feature set sizes. Non-exhaustive methods need not examine every subset, but the different methods vary in the number of subsets they do need to examine in order to find the optimal set. For

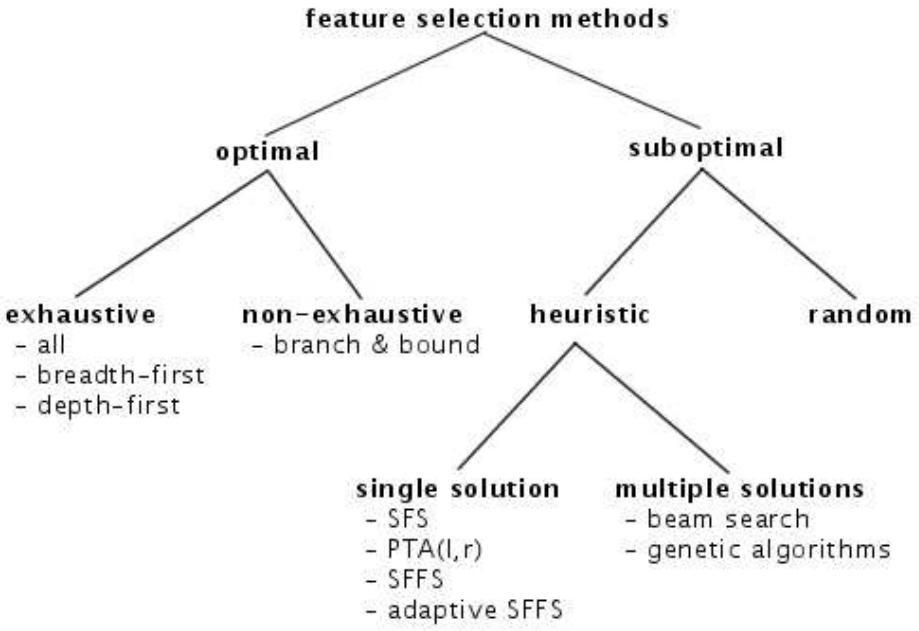

Figure 3.1: A taxonomy of feature selection methods.

instance, Branch and Bound is faster that depth-first search, which in turn is faster than breadth-first search when confronted with cases of exponential growth of the search space. For a detailed explanation of these algorithms, the reader is referred to [Winston, 1992]. However, the Branch and Bound method is impractical for problems with very large feature sets, because the worst case complexity of this algorithm is exponential. A number of other non-exhaustive methods exist, but these are not relevant here.

It is important to note that the promise of finding the optimal set comes with a strict requirement that is shared by all the non-exhaustive feature selection methods: walking through the search tree requires a monotonic evaluation function. This means that the addition of new features to a feature subset can never decrease the value of the evaluation function.

3.2.2 Suboptimal

The suboptimal methods are further divided into those use heuristics of some kind to find a suboptimal feature set and those that randomly search the feature subset space.

Heuristic

Heuristic methods can be divided into those that store just one current feature subset and make modifications to it, versus those that maintain a population of subsets. Another possible distinction can be made between deterministic algorithms, which produce the same subset on a given

problem every time, and algorithms that have a random element which could produce different subsets on every run.

The first and most common group of heuristic methods begin with a single solution in the form of a subset of features and iteratively add or remove features until the stopping criterion is met. In the first case, the method starts with an empty set and and iteratively adds features to it. These methods are called *forward* or *bottom-up* methods, while the so-called *backward* or *top-down* methods start with the full set and delete features. It depends on the method itself whether or not features can be deleted or added in the forward and backward methods respectively.

At first glance there does not appear to be much difference between these two approaches, but backward feature selection does have a cumbersome disadvantage: calculating the values of the evaluation function is generally faster when the feature set is smaller. This is especially the case when the wrapper approach is used, which means that, generally speaking, backward selection is slower than forward selection.

One of the oldest feature selection methods around is *Sequential Forward Selection* (SFS) which starts the search with an empty set. Each iteration all the variables that have not yet been selected are temporarily added and their impact on the value of the evaluation function is recorded. The feature whose inclusion resulted in the best score is permanently added to the set. This process is repeated until a pre-specified number of iterations has been reached or until there are no variables left whose inclusion improves the evaluation score. The counterpart of SFS, *Sequential Backward Selection* (SBS), removes the worst feature relative to the evaluation function each iteration until a similar stopping criterion has been met.

The SFS and SBS methods both suffer from the so-called nesting property, which is often seen as a drawback in feature selection literature [Jain and Zongker, 1997]. This nesting means that, in the case of SFS, once a variable is included, it cannot be excluded later, even if it might be possible to increase the evaluation score by doing so.

Several solutions have been proposed over the years to remedy this problem, one of which is know as the *plus l - take away r* or PTA(l,r) method. PTA(l,r) involves applying l steps of SFS followed by r steps of SBS until a desired number of features has been reached. Even though the problem of nesting can be partially alleviated with this approach, there is no good way of predicting the best values of l and r [Ferri et al., 1994].

A decent solution to this problem was not presented until 1994 when Pudil et al. proposed the *Sequential Forward Floating Selection* (SFFS) method [Pudil et al., 1994]. The way that SFFS tries to counteract the nesting problem is by using backtracking after each round of SFS. Each iteration, one round of SFS takes place and the best feature is included. This step is followed by a backtracking phase that looks for features whose exclusion will improve the evaluation score and this exclusion is carried on for as long as better feature subsets are found. When no better subset is found, the algorithm goes back to the first SFS step and includes the best not yet

excluded feature, which is again followed by the backtracking phase. The *Sequential Backward Floating Selection* (SBFS) algorithm is similar to SFFS but performs the inclusion and exclusion steps in reversed order. SFFS and SBFS have in many comparisons proven to be superior to the SFS/SBS and PTA(l,r) algorithms even though they are more complex and the search takes more time [Jain and Zongker, 1997, Jain et al., 2000, Reunanen, 2003].

Somol et al. proposed an improved version of the floating search methods called adaptive floating search (or ASFFS/ASBFS) [Somol et al., 1999]. These methods are adaptive in the sense that the number of features that are included or excluded in each iteration is determined dynamically. Research has shown that ASFFS usually outperforms SFFS albeit that the improvement is sometimes only marginal and at the cost of increased execution time [Reunanen, 2003].

The second category of heuristic methods are those that maintain a queue or population of possible subsets. The most popular multiple solution method involves the use of genetic algorithms (GA) for feature selection and was pioneered by Siedlecki and Sklansky [15]. In the typical GA approach, any given feature subset is represented as a binary string (or a chromosome) of length n, with a zero or one in position i denoting the absence or presence of feature i in the set where n is the total number of available features. A population of these chromosomes is maintained and each chromosome is evaluated to determine its fitness, which determines how likely the chromosome is to survive and breed into the next generation. New chromosomes can be created in two different ways: by crossover, where parts of two different parent chromosomes are mixed to create offspring, and by mutation, where the bits of a single parent are randomly flipped to create a mutated child.

Another, less popular feature selection algorithm that maintains a queue of multiple solutions is beam search, but it also requires a monotonic evaluation function and is therefore not relevant for our NER feature selection problem. For a description of the algorithm, the reader is referred to [Winston, 1992].

Random

Like the heuristic suboptimal feature selection methods, methods that randomly search the feature subset space are not guaranteed to find the optimal feature set. Random methods always require a stopping criterion that stops the feature selection method when a predefined number of iterations has been reached since there is no way of knowing whether or not the current best set is the optimal set unless an exhaustive random search is performed.

3.3 Feature selection and algorithm parameter optimization

As mentioned before in this chapter, feature selection can provide a better generalization performance due to the exclusion of irrelevant or redundant features. However, feature selection is but one of many factors that can influence the outcome of machine learning experiments: the original data used, the feature representations and the parameter settings of the algorithms used also play

an important role in the outcome. Optimization of the latter often increases the generalization performance considerably as well.

However, these different factors are not independent from one another. For instance, the feature selection process is obviously highly dependent on the original data and Daelemans et al. showed in their 2003 paper that feature selection and algorithm parameter optimization are also closely intertwined [Daelemans et al., 2003a]. They argued convincingly that feature selection, algorithm parameter optimization, and their joint optimization cause larger differences in generalization performance within a single algorithm than differences observed between different algorithms.

This implies that determining the best possible feature set and parameter settings is an even more extensive process than separate feature selection and parameter optimization would be as this joint optimization is of a combinatorially explosive nature. Daelemans' successful solution to this optimization problem in large search spaces are genetic algorithms. However, a big disadvantage of GAs is that they are computationally very expensive. We will present our own solution to this problem in the next chapter about the experimental setup.

Chapter 4

Experimental setup

In this chapter the setup of the experiments used to answer the research questions will be described. First the original dataset will be examined, followed by a description of the feature extraction and selection process. The algorithms used will be discussed in the third section and the post-processing steps will be highlighted in the final section of this chapter.

4.1 Data

In order to answer the research questions posed in chapter 1 we used the Dutch datasets provided for the 2002 CoNLL shared task for the NER experiments. The Dutch datasets consist of four editions of the Belgian newspaper 'De Morgen' of 2000 (June 2, July 1, August 1 and September 1) and the data was annotated as a part of the Atranos project at the University of Antwerp [ESAT/PSI speech group, 2004].

The data consist of three columns separated by a single space and each word has been put on a separate line. Sentence breaks are encoded by empty lines. The first item on each line is a word, the second the part-of-speech tag and the third item the named entity tag. There are 14 different part-of-speech tags and they were generated using (an older version of) the MBT tagger [Daelemans et al., 2003b] and checked for correctness. The named entity tags consist of two parts separated by a dash. The first part is a chunking tag, according to the IOB2 chunk tagging scheme [Veenstra and Tjong Kim Sang, 1999]. In this scheme a `B` denotes the first item of a named entity phrase and an `I` any non-initial word in a phrase. An `O` denotes all tokens that are not part of a named entity phrase. The second part of the named entity tag determines the type of phrase: person names (`PER`), organizations (`ORG`), locations (`LOC`) and miscellaneous names (`MISC`). This means that there are a total of 9 different named entity tags (`B-LOC`, `I-LOC`, `B-ORG`, `I-ORG`, `B-MISC`, `I-MISC`, `B-PER`, `I-PER`, and `O`). Named entities do not cross over sentence boundaries and they also do not overlap. Article boundaries have been marked by a special tag (`-DOCSTART-`). Example 6

shows the first sentence of one the articles from the data and its CoNLL representation is shown below the example.

(6) Dat is in Italië , Spanje of Engeland misschien geen probleem, maar volgens ' Der Kaiser ' in Duitsland wel.

```
-DOCSTART- -DOCSTART- O
Dat Pron O
is V O
in Prep O
Itali N B-LOC
, Punc O
Spanje N B-LOC
of Conj O
Engeland N B-LOC
misschien Adv O
geen Pron O
probleem N O
, Punc O
maar Conj O
volgens Prep O
' Punc O
Der N B-PER
Kaiser N I-PER
' Punc O
in Prep O
Duitsland N B-LOC
wel Adv O
. Punc O
```

As mentioned before, the data were split into a training set, a development set, and a test set and distributed to all participants who proceeded by training their learning methods on the training set. The development data was available to help participants tune the parameters of their learning methods. When the best parameters had been found, the resulting NER system would be trained on the training data and tested on the test data. Table 4.1 gives an overview of some of the numeric data characteristics of the three datasets.

4.2 Features

In order to maximize the generalization performance as stated in the first research question, the first step is determining the best indicators of the named entity type from a pool of potentially useful features. This process was split up into two stages. First, a large number of potentially useful features were extracted from the words in the datasets. Using a large number of features

	training set	development set	test set
no. of articles	287	74	119
no. of sentences	15,806	2,895	5,195
no. of instances	202,931	37,761	68,994
no. of named entities	13,344	2,616	3,941
no. of unique named entities	5,713	1,267	1,864
average length named entities in words	1.446	1.420	1.461
total class entropy	0.717	0.738	0.646
entropy within four NE types	1.943	1.981	1.980

Table 4.1: Some statistics of the Dutch CoNLL-2002 datasets.

certainly does not guarantee optimal generalization performance because many features can turn out to be partially or completely irrelevant, or redundant. This is why feature selection is such an important step in finding the sub-optimal feature set: this was the next stage in maximizing the generalization performance.

4.2.1 Feature extraction

A large number of features were extracted from the datasets and many of these were inspired by the best approaches discussed in chapter 2. This pool was completed with several other ideas for potentially useful features, leading to the following set of features that were extracted from each instance in the datasets:

CoNLL features These features were provided by CoNLL in the original datasets.

- ▷ original word
- ▷ part-of-speech tag; the original POS tag as predicted by the MBT tagger and corrected later on

Orthographic features These are non-exclusive, binary features that test whether or not the following predicates about the orthography of a word hold. By combining these predicates any kind of orthographic characteristic can be represented.

- ▷ `firstCap`; does the word starts with a capital letter?
- ▷ `allCaps`; is the whole word in upper case?
- ▷ `internalCaps`; does the word have internal upper case letters (not counting the first letter)?
- ▷ `allLowercase`; is the whole word in lower case?
- ▷ `containsDigit`; does the word contain one or more digits?
- ▷ `containsDigitAndAlpha`; does the word contain both digits and alphanumeric characters?

- ▷ `onlyDigits`; is the word made up entirely of digits? (this does not hold for, for instance, "30,000")
- ▷ `isPunctuation`; is the word made up entirely of punctuation marks?
- ▷ `containsPunctuation`; does the word contain any punctuation?
- ▷ `isHyphenated`; is the word hyphenated? (the hyphen has to be inside the word, that is not the first or last character of a word)

Word type features These binary features represent whether or not a word is of a specific type.

- ▷ `firstSentenceWord`; is the word the first word of a sentence?
- ▷ `isInitial`; does the word represent an initial of some sort? (single letter initials are supported as well as expressions like "A.B." or "AB." of up to 5 letters)
- ▷ `quotedText`; is the word quoted text, that is, enclosed between a pair of single or a pair of double quotes? (note that the quote characters are also considered to be (part of the) quoted text; also note that a sentence boundary automatically ends any quoted text)
- ▷ `isURL`; is the word a URL? (a word is considered to be a URL if it starts with "http", followed by either a ":" or a "/")
- ▷ `functionWord`; is the word a function word (or stopword), in other does the word occur in the list of Dutch function words? This list was a combination of the function words listed in [Bagola, 2003] and the words with an adverb POS tag (`Adv`) extracted from the CoNLL training dataset.

Statistical features These features represent certain statistical characteristics of a word.

- ▷ `wordLength`; the number of characters in the word.

Morphological features These features represent the different morphological characteristics of a word

- ▷ `prefix`; the first 4 letters of the word (we decided against extracting precise morphological information from a specialized database such as Celex because it was computationally very expensive and the gains in generalization performance were considered to be minimal)
- ▷ `suffix`; the last 4 letters of the word

Output feature This nine-valued feature is the output class and represents the type of named entity of a word.

Simply extracting these features for the focus word and using them to predict its NE class would not yield the best results; section 2.1 mentions a number of different examples where the context can help to classify a named entity. Therefore, context needs to be added in order to be recognized these kinds of linguistic patterns and the standard way of representing the local context is by anchoring a window in the focus word w. Features are extracted for each of the

words in the window around w and fed to the classifier to predict the output class of w. This process automatically includes information about the relative position of the context words to w [Carreras et al., 2002].

We made the decision to include two words to the left and to the right of the focus word as the context (2-1-2) and extracted the aforementioned 20 features for each of the five words. This yielded a total of 101 features (including the output class) for each instance. The 2-1-2 window size was selected, because it is a popular window size in the literature [Wu et al., 2002, Hendrickx and Van den Bosch, 2003, Bender et al., 2003]. Windows of 6 context (3-1-3) are also popular, however, using a larger context window such as 3-1-3 would have resulted in 40 extra features and feature selection is a time-consuming process. Using a smaller context window was not advisable either since restricting a classifier to patterns of three words or less would not be sufficient to cover an adequate portion of all named entities; several named entities consist of more than four words.

A common approach is to include seedlist information by encoding their occurrence in a specific seedlist as a binary feature. We decided against including seedlist features in the datasets before the feature selection process so that their influence on generalization performance can be measured more precisely. Section 4.5 contains more information about how seedlists were incorporated.

Each of the three datasets—training, development and test—was transformed into a windowed dataset of instances with 100 features and 1 output class. Sentence boundaries and other empty places in the context windows were marked by a special tag (`-EMPTY-`), analogous to the article boundary tag (`-DOCSTART-`)[1]. Appendix A contains an example of four instances from one of these transformed datasets.

4.2.2 Feature selection

Feature selection can provide a better generalization performance due to the exclusion of irrelevant or redundant features, but it was argued in the previous chapter and convincingly by [Daelemans et al., 2003a] that this is not independent of the optimization of the parameter settings of the learning algorithms. One successful solution would be to use genetic algorithms to handle the large search spaces. However, this is not practical for the experiments in this thesis due to the computationally expensive nature of GAs.

We propose a compromise between independent feature selection and parameter optimization and an interleaved solution. We have used two different machine learning algorithms in our experiments, MAXIMUM ENTROPY (or MAXENT) and k-NEAREST NEIGHBOR (or k-NN), and the former has virtually no parameters that need to be set. Therefore, the default parameter settings of 'our' implementation of the maximum entropy algorithm were used and kept constant throughout the feature selection process. This resulted in a virtually completely interleaved feature selection and parameter optimization process for the MAXENT learner.

[1] Feature extraction was implemented as a PHP script and ran on a Pentium 4 2.0 GHz with 512 MB of RAM running SUSE 9.0.

Unfortunately, the k-NN algorithm has many different parameters and most of them can have a profound influence on the generalization performance by themselves. Combining this with an extensive feature selection phase would lead to an unacceptably large search space, even for memory-based learners. Because of time and computational considerations a compromise was made: we assumed that the same feature set that was sub-optimal for the MAXENT algorithm would be a good approximation of the sub-optimal feature set for the k-NN algorithm. This compromise saved valuable time by only requiring feature selection for the MAXENT algorithm and parameter optimization for the k-NN algorithm. The resulting feature set is part of the answer to the first research question.

The feature selection method that was used in all the experiments was the *Sequential Forward Floating Selection* (SFFS) method [Pudil et al., 1994] because of the advantages discussed in section 3.2.2. The disadvantages of the adaptive version of this floating search method (ASFFS), such as increased execution time, were deemed too great to prefer it to SFFS[2]. A wrapper approach was used by employing the MAXENT algorithm as the evaluation function. The reason for using a wrapper approach and not a filter approach is that the MAXENT algorithm is likely to apply some form of filtering itself in the learning process. The wrapper approach is likely to be closer to the end result since the same inductive algorithm is used there. The SFFS heuristic itself was implemented as a PHP script; PHP may not be the fastest or most efficient scripting language, but the computational bottleneck was the evaluation function, so this did not really present any problems.

4.2.3 One-step recognition vs. separated chunking and classification

The feature selection process is also essential in answering the second research question of whether the machine learning algorithms should be trained to predict the named entity boundaries and their types at the same time or whether it is preferable to first identify the boundaries and then classify the named entities in a separate step.

The one-step option can be examined by simply running the feature selection process on the current windowed datasets. The MAXENT algorithm will then be used as the evaluation function to predict the original nine-valued named entity tags. The sub-optimal feature set discovered by SFFS will be discussed in the next chapter.

For the two-step option, first the named entity boundaries need to be predicted. This means a another version of the original, windowed dataset was created where the nine-valued named entity tags were replaced by three-valued tags signaling the beginning, interior or exterior of a named entity (`B`, `I`, or `O`). Feature selection on this modified dataset yielded the sub-optimal set of features that best predicted the named entity chunks. In the second stage of classifying the newly chunked named entities, the original, windowed dataset of 100 features is then supplemented with one extra feature: the boundaries of the named entities.

[2]The feature selection experiments were run on a dual CPU computer (each CPU a Athlon 2.2 GHz) with 1 GB of RAM running Slackware 8.0.

This extra feature presents us with a new dilemma since it can be constructed in two different ways. The first is using the results of the chunking stage as (imperfect) predictions; the second is distilling them from the original nine-valued named entity tags as was done in the chunking stage. These chunking predictions (perfect or imperfect) were added as an extra feature after the feature selection process. Using these perfect chunks gives us a ceiling performance with which to compare the performance of relying on the predictions of the chunking classifier [3].

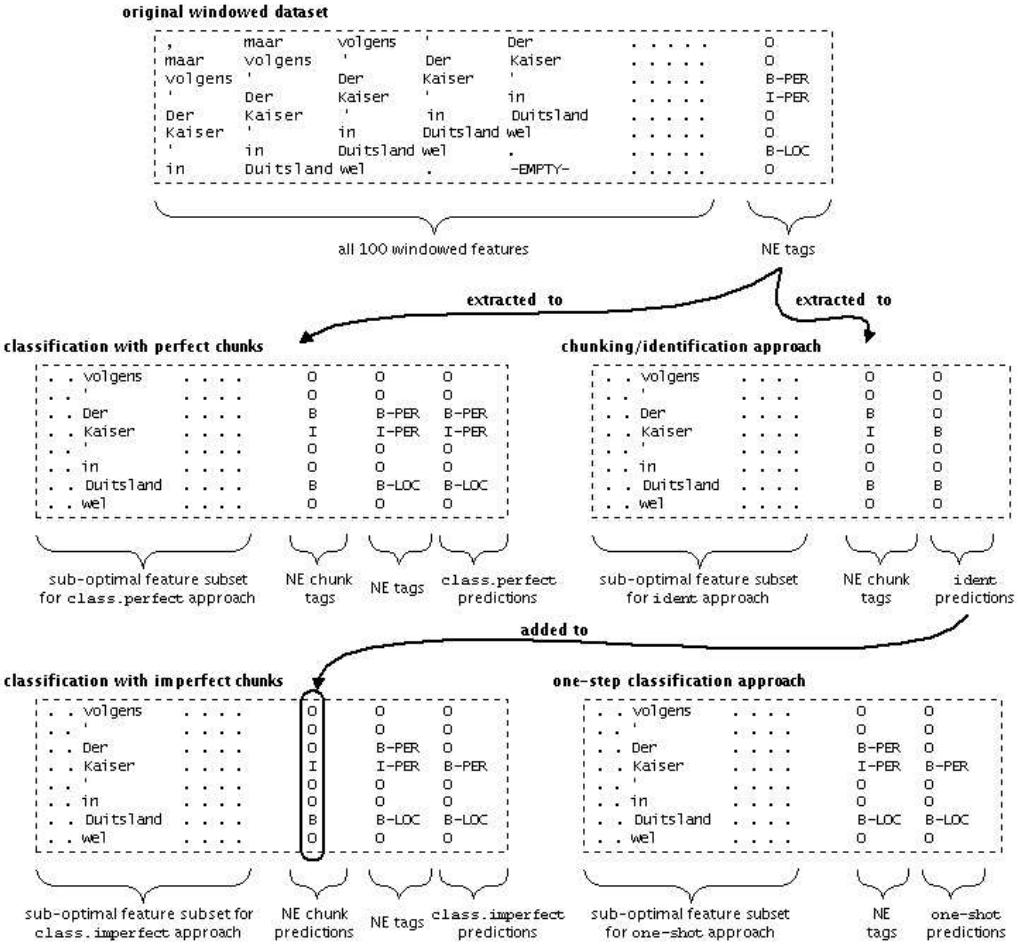

Figure 4.1: An overview of the four different feature selection problems and their relation to one another.

All of this means that a total of four different feature selection experiments were performed, resulting in four different feature sets, each sub-optimal for its own prediction problem. Figure 4.1 clarifies the process for the four datasets: one-step classification (`one-shot`), chunking/identification

[3]Obviously, the only way of classifying sentences that are not in the three datasets always involves using the imperfect chunks, because the perfect chunks are not known.

(`ident`), classification with imperfect chunks (`class.imperfect`), and classification with perfect chunks (`class.perfect`).

In each of the four feature selection experiments, the following four steps were taken. First, the current feature subset according to SFFS was created. We decided to test the evaluation function only on the training data by performing 10-fold cross-validation. According to [Kohavi, 1995] the process of k-fold cross-validation involves randomly splitting the data set D into k mutually exclusive subsets (or folds) $D_1, D_2, \ldots D_k$ of approximately the same size. The machine learning algorithm is then trained k times; each time $t \in 1, 2, \ldots, k$ it is trained on $D \backslash D_t$ and used to predict the labels of D_t. The predictions of all 10 folds were combined and evaluated using the evaluation script provided by the organization behind the CoNLL 2002 shared task. The accuracy of the learning algorithm is then defined as the average of the classification F-scores on the k different test sets. Note that the test dataset was not used for any kind of evaluation in any of the experiments: to make the approaches of all the participants in a shared task comparable, the test data may only be used for the final evaluation.

The training dataset was split into 10 training folds and 10 corresponding test folds using a Perl script. The MAXENT algorithm requires a dataset with only binary features, so each of the 10 train and test folds was binarized using another Perl script. The MAXENT algorithm was trained on a training fold and the resulting classifier was used to predict the class labels of the test folds. The aforementioned evaluation script was used to determine the F-score of each of the ten folds. The average F-score of the 10 folds was used as the evaluation function measure in the SFFS method. Note that the presence of the five windowed words as well as the five windowed part-of-speech tags in the feature subsets was fixed; they could not be removed from the set of useful features. This was done partly because of the undesired increase in execution time of the feature selection process and because of their presupposed importance in the classification process. The four feature subsets that resulted from these feature selection experiments are described in detail in the next chapter in section 5.1.

4.3 Algorithms

The next step in answering the first research question is determining what algorithms and algorithm parameters should be used to maximize the generalization performance of Dutch NER. We used two algorithms that are known to perform well on IE extraction tasks: MAXENT and and k-NEAREST NEIGHBOR, both of which will be described in more detail in the next two subsections. As mentioned before, the MAXENT algorithm has virtually no parameters that need to be set. Parameter optimization of the k-NEAREST NEIGHBOR algorithm was done using a program called PARAMSEARCH. It employs a heuristic meta-learning search method called Wrapped Progressive Sampling (WPS). The basic principle behind WPS involves testing a decreasing number of parameter settings on datasets of increasing size. The method starts with a small training and test set and trains a machine learning algorithm on the training set for every possible combination of parameter settings. The portion of the settings that performs best at predicting the class labels of test set is kept and the dataset sizes are increased. The same procedure is once again performed

on the training and test set using the remaining combinations of parameter settings. Once again the best-performing settings are kept and the dataset size is increased and so on until there is only one setting left or until the dataset size cannot be increased any further. In the latter case a random setting is selected (or the default setting if it is a member of the set of remaining parameter settings) [Van den Bosch, 2004].

4.3.1 Maximum entropy

Maximum entropy modeling is a general purpose machine learning framework that has proved to be highly expressive and powerful in statistical natural language processing, statistical physics, computer vision and many other fields. MAXENT models are a class of statistical models which require no unwarranted independence assumptions. They hinge on the notion of modeling all that is known and assuming nothing about that which is unknown [Malouf, 2002b]. This makes MAXENT a typical greedy learner in that it first models the training data and then classifies query instances using that model.

Modeling all that is known is done by constructing a statistical model of the process which generated the training sample. The building blocks of this model will be a set of statistics of the training sample. These statistics could be both dependent (4.2) on and independent (4.1) of the context x:

$$f(x,y) = \begin{cases} 1 & \text{if } y = \texttt{B}-\texttt{PER} \ \& \ \text{focus word} = \textit{Javier} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

$$f(x,y) = \begin{cases} 1 & \text{if } y = \texttt{I}-\texttt{LOC} \ \& \ \text{previous tag} = \texttt{B}-\texttt{LOC} \\ & \ \& \ \text{previous word} = \textit{New} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Any statistic of the sample can be represented as a binary-valued indicator function f and the expected value of this f with respect to the empirical distribution $\tilde{p}(x,y)$ is important. Any important statistics are acknowledged by requiring them to accord with the model of the process that generated the training sample. This is done by constraining the expecting value in this model, $p(x,y)$, to be the same as the expected value in the training sample, $\tilde{p}(x,y)$, as in 4.3 [Berger et al., 1996].

$$p(x,y) = \tilde{p}(x,y) \tag{4.3}$$

If we are given n different feature functions f_i that represent statistics that are important in modeling the process, then we want our model of the process to accord with all of these statistics in the training sample. In other words, we require our model p to lie in the subset C of the set of all probability distributions P as in:

$$C \equiv \Big\{ p \in P \mid p(x,y) = \tilde{p}(x,y) \quad \text{for } i \in \{1,2,\ldots,n\} \Big\} \tag{4.4}$$

The difficulty lies in selecting the best model p from the set C of allowed probability distributions. The maximum entropy principle dictates that we should assume nothing about that which is unknown. That is, the optimal model p_* should be as uniform as possible and a well-known mathematical measure of the uniformity of a conditional distribution $p(y \mid x)$ is provided by the conditional entropy:

$$H(p) \equiv -\sum_{x,\,y} \tilde{p}(x)p(x \mid y) \log p(x \mid y) \tag{4.5}$$

This means that, taking into account the important statistics of the training sample, the most uniform and least biased model is the model with the maximum entropy (4.6). It can be shown that there is always a unique model p_* with maximum entropy in any constrained set C and different methods exist of converging to this model such as generalized iterative scaling (GIS), improved iterative scaling (IIS), and a quasi-Newton unconstrained optimization algorithm called L-BFGS. The reader is referred to [Berger et al., 1996], [Ratnaparkhi, 1997] and [Malouf, 2002a] for the relevant proofs and discussion.

$$p_* = \underset{p \in C}{\operatorname{argmax}}\, H(p) \tag{4.6}$$

One of the advantages of the MAXENT approach for named entity recognition is that this problem, like many information extraction problems is categorized by a large number of weakly predictive features which will almost always have certain exceptions. Taking the product of the features' weightings to see which future is predicted most strongly thus seems like a reasonable thing to do in such a situation. Another factor working in favor of maximum entropy for NER is that it only has to choose from a very limited range of tags (9) for each word. A wider future space would have made it much more computationally intensive [Borthwick, 1999]. For these reasons, and the fact that MAXENT seemed to perform well in the literature, we chose MAXENT as one of our two classification algorithms.

A number of different maximum entropy implementations exist and a short overview can be found in [Le, 2004]. We used Zhang Le's MAXENT implementation, version *20040315*, which is a MAXENT modeling toolkit written in C++ and Python. It offers several different parameter estimation methods such as GIS and L-BFGS. The parameter settings we used in the feature selection are the default 30 iterations for the default training algorithm of L-BFGS. These and all other parameter settings were all default settings of the MAXENT implementation.

4.3.2 Memory-based learning

In contrast to MAXENT modeling where a model is derived from the training data to use in classifying query instances, memory-based learners do not construct an explicit model: they simply store all the training examples and refrain from making any abstractions. When a new query instance is encountered, a set of instances similar to the query is retrieved from memory and used

to classify the query instance. An implementation of the popular k-NEAREST NEIGHBOR (or k-NN) algorithm was used for our experiments, but memory-based learning also includes locally weighted regression and case-based reasoning [Mitchell, 1997].

The k-NN algorithm hinges on the assumption that all instances correspond to points in an n-dimensional feature space F^n and that a new query instance q is best classified by selecting the k instances that are closest to q in the feature space and thus the most similar to q. The simplest method for computing the similarity between two instances x and y is the *simple overlap metric* where the similarity $\delta(x, y)$ between x and y is computed as the number of matching feature values of x and y. Other distance metrics include the *dot product* and the *modified value difference metric*. In the dot product metric the distance between two instances is calculated by subtracting the dot product of the feature value vectors from the maximum dot product (an exact match). The modified value difference metric (or MVDM) is a method to determine the similarity of the values of a feature by looking at co-occurrence of values with target classes, since some feature values are more similar to each other than to others, in that they occur often with the same class(es). The *Jefferson divergence metric* is similar to MVDM and is used to compute the distance between class distributions of two values of the same feature. In theory, the Jefferson divergence metric is more robust to sparse data [Daelemans et al., 2003c].

The use of these metrics can be refined by using *feature relevance weighting* which acknowledges that not every feature is as equally important for predicting the class of an instance. Therefore, weighting metrics are used to reflect each feature's predictive power, such as *information gain*, *gain ratio*, *Chi-squared* (χ^2) *weighting* and *shared variance weighting* [Dake, 2003, Daelemans et al., 2003c]. The information gain of a feature x_i represents how much knowing the value of x_i decreases the uncertainty about the class. Gain ratio alleviates a potential drawback of information gain—it favors features with many different values which can lead to problems—by dividing the information gain by the entropy of a feature. Unfortunately, the gain ratio measure still has an unwanted bias towards features with more values, because it is not corrected for the number of degrees of freedom of the contingency table of classes and values. Chi-squared weighting addresses this problem and uses the χ^2 values as feature weights. Shared variance weighting is similar to Chi-squared weighting except it explicitly corrects for the degrees of freedom [Daelemans et al., 2003c].

If k is not equal to 1 a choice has to be made how to weigh the input of the k different nearest neighbors in choosing the class of the query instance. A number of different selection methods exist to address this problem and the most straightforward of these is *majority voting* where each of the k nearest instances contribute equally to the classification decision. It can be argued successfully that instances that are closer to the query instance q should have a larger influence on the classification decision and *distance weighted class voting* methods address this issue. A successful voting scheme uses weights that are a functions the distance between q and the nearest instances. In the *inverse linear weighting* scheme this is done by assigning the nearest neighbor a weight of 1, the furthest neighbor a weight of 0 and scaling the other weights linearly to the interval in between. The *inverse distance weighting* scheme calculates the voting weights by inverting the distance between q and one of the nearest instances. Another weighting function, *exponential decay*

weighting, is based on Shepard's universal perceptual law which states that the relevance of a previous stimulus for the generalization to a new stimulus is an exponentially decreasing function of its distance in a psychological space [Daelemans et al., 2003c].

We chose the k-NN algorithm as our second algorithm because it has been successfully applied to many different NLP tasks and because of the expertise regarding memory-based learners at ILK [ILK, 2004]. For our k-NN experiments we used the memory-based learner implementation TIMBL, version 5.0 patch 2 [Daelemans et al., 2003c]. It is written in C++ and supports a number of other classification algorithm in addition to k-NN such as IGTREE, decision-tree-based optimization and hybrid and non-parameterized version of these respective algorithms. It supports all of the distance metrics and feature-weighting possibilities described above as well as the four voting methods.

Because the performance of the k-NN algorithm crucially depends on the choice of metric, feature-weighting and voting method, the PARAMSEARCH program was used on the complete training dataset to perform the necessary parameter optimization steps. It then produced a file containing the best settings for TIMBL to set the parameters of the k-NN algorithm.

4.4 Basic experiments

The first round of experiments were performed on the datasets containing the five windowed words, the five windowed part-of-speech tags and only the subsets of features found by the feature selection rounds for each of the following four tasks: one-step classification (`one-shot`), chunking/identification (`ident`), classification with imperfect chunks (`class.imperfect`), and classification with perfect chunks (`class.perfect`). These reductions of the original windowed datasets to the suboptimal datasets were done for the training dataset as well as the development and test dataset.

In addition to these four experiments, we also trained MAXENT and TIMBL on windowed datasets containing all of the 100 features (`all`) to examine whether or not the feature selection process actually improved generalization performance[4].

For each of these five experiments, both algorithms were trained on the training dataset and tested on the development and test sets. The MAXENT algorithm required that the datasets be binarized before training using the binarization script mentioned in 4.2.3. The PARAMSEARCH program was used to determine the optimal TIMBL settings for classification. The results of these experiments can be found in section 5.2. The F-scores of these experiments (and of the other experiments mentioned further on) were calculated using the evaluation script provided by the organization behind the CoNLL 2002 shared task.

Note that an extra feature column of chunk tags was added to the `class.imperfect` and `class.perfect`. The chunking tags added to the 'perfect' dataset were extracted from the origi-

[4]These five experiments (and the experiments in the following sections as well) were performed on were run on a dual CPU computer (each CPU a Athlon 2.2 GHz) with 1 GB of RAM running Slackware 8.0.

nal nine-valued named entity tags (as was done in the chunking stage). The chunking tags added to the 'imperfect' datasets were the predictions on the `ident` dataset and this resulted in two different versions of these datasets (`class.imperfect.maxent` and `class.imperfect.timbl`), because two different classifier algorithms were used to predict the chunks. The 'imperfect' chunks also had to be added to the training sets of the `class.imperfect` tasks; we used 10-fold cross-validation to predict the chunk tags on the `ident` training set for use in the `class.imperfect.timbl` and `class.imperfect.maxent` training sets.

4.5 Seedlists

The goal of the second round of experiments was to measure whether or not seedlist features are helpful in recognizing named entities. As mentioned in section 4.2.1, seedlist features were added to the datasets after the feature selection stage so that their influence on generalization performance could be measured more precisely.

Seedlist information was included by encoding a word's occurrence in a specific seedlist as a binary feature. We used four seedlists, one for each type of named entity, and the lists contained the complete expressions. This means that a named entity such as "Niagara Falls" is stored as such in the seedlist and not as the two separate words "Niagara" and "Falls". The seedlists were a combination of named entities extracted from the training set and sorted by type and named entities collected from various Internet sources. The following table shows how many named entities each of the four seedlists contained:

NE type	number of items
LOC	6,666
ORG	7,758
MISC	7,005
PER	109,165

A PHP script was used to create the 4 seedlist feature column and only the longest possible matches were used. This means that if the words "Santiago de Compostela" occurred in the dataset, they would be tagged as a three-word NE as long as the location seedlist contained the named entity "Santiago de Compostela". Even if the seedlist contained the shorter "Santiago" as well, it would still tag the longer version.

This round of experiments was performed on the datasets used in the basic experiments, supplemented with four binary features for the four types (`LOC`, `ORG`, `MISC`, and `PER`). Once again, these experiments were performed for the six different datasets (`one-shot`, `ident`, `class.perfect`, `class.imperfect.maxent`, `class.imperfect.timbl`, and `all`).

Once again, both algorithms were trained on the training dataset and tested on the development and test sets for each of these six experiments. The datasets were binarized for the MAXENT algorithm and the PARAMSEARCH program was used to determine the optimal TIMBL settings. The 'imperfect' chunk tags added to both `class.imperfect` datasets were the best predictions on

the `ident` task made in the two rounds. This means that if adding seedlist features to the dataset of the chunking task in the seedlist round did not result in any performance gains, the best chunking results so far were used. This procedure was also followed in the next round of experiments.

The first version of the binarization script handled features that were already binary in the dataset by creating two new binary features: one for the 'on'-values and one for the 'off'-values. This means that a binary feature was transformed into two new binary features, one the exact opposite of the other. This did not present any problems in the feature selection stage, because selecting one of these new' features automatically ensures that including its counterpart as well would yield absolutely no improvement.

For the experiment rounds we examined whether or not this binarization behavior yielded worse results than leaving the binary features intact. However, MAXENT performed better on most of the datasets when they were binarized using the original script that doubled the number of the binary features so this original script was used. The results of these experiments can be found in section 5.3.

4.6 Classifier stacking

The third round of experiments focused on the possibility of improving classifier performance by using second-stage stacking. Stacking is described in more detail in section 2.2.1 and involves combining the predictions from multiple classifier models. In second-stage stacking, the results of the previous classification round are added as windowed features to the instances presented to the second classifier. In theory, the second-stage classifier should be able to learn to recover from recurring errors made by the first-stage classifier [Veenstra, 1998].

We deviated slightly from the approach taken by [Hendrickx and Van den Bosch, 2003] in that the windowed features that were added to the instances in the training set were not the predictions of the first-stage classifier, but a window of the actual named entity tags as provided by CoNLL. Obviously, this means that the tag of the focus word could not be included because this would have biased the classifiers into making perfect predictions using only that feature. This means that four features were added to the datasets: the original named entity tags of the four words surrounding the focus word were added to the training set, and the predictions of the best-performing classifier on the same task of the previous rounds. This particular approach was also used by [Van den Bosch et al., 2004] for their memory-based learning approach to semantic role-labeling[5].

One of the drawbacks of this approach is that it could introduce unexpected classification errors by training a classifier on perfect 'predictions' and not on the same 'flawed' predictions that both test sets contain. Although this could result in discrepancies, this was taken for granted since this approach is faster than using predictions in all three datasets. Furthermore, both approaches were tested on different datasets and the 'perfect' approach turned out to perform better on En-

[5]The 2004 CoNLL shared task.

glish NER data.

This round of experiments was performed only on the 'best-performing' combinations of algorithms and datasets. This was done because it was clear that several of the other combinations were clearly lagging behind compared to the best approaches and the goal was to determine which single combination yielded the best generalization performance. In addition, performing second-stage stacking on all of the datasets would have taken too much time.

Two datasets were selected for the third round of experiments: the best combination of dataset and algorithm for the one-step approach and the best combination for the two-step approach. The windowed prediction features were added to these two datasets (`class.perfect.timbl` and `all`) and `Timbl` was trained on the training dataset and tested on the development and test sets for each of these two approaches. Note that we selected the ceiling performance of the perfect chunking approach as the best two-step approach because our aim was to maximize the generalization performance on the CoNLL 2002 shared task, not on sentences that were not in these three datasets. This ceiling performance also makes for interesting comparisons with other imperfect two-step approaches. Once again, the PARAMSEARCH program was used to determine the optimal TIMBL settings. The results of these experiments can be found in section 5.4.

4.7 Post-processing

Our goal was to maximize generalization performance on the NER task so we decided to see if we could improve upon the performance of the classifier by employing a hybrid approach. The best classifier was selected on the basis of the test scores and the influence of post-processing rules was measured on the results of this classifier. We also applied the rules to the ceiling performance of the perfect chunks task to determine how much other two-step approaches have to make up for. An analysis was made of the most frequent errors made by the best-performing classifier and rules were crafted by hand to correct these errors similar to [Tjong Kim Sang, 2003]. Extra information was extracted from the datasets used in the post-processing experiments: the predictions of the classifier stacking round were windowed (2-1-2) and the chunks and types were extracted from the predictions and windowed as well. For instance, if the classifier predicted the tag sequence `O B-LOC I-LOC O O` then the following features were added to the instance representing the focus word of that tag sequence: `O B-LOC I-LOC O O` `O B I O O` `O LOC LOC O O`. Despite the obvious redundancy of the last 10 of these features, they are not fed to a classifier in this round of experiments but to a set of rules which means that classification performance can never be decreased (if the right rules are selected).

Two different kinds of rules were used: rules that correct impossible tag combinations and rules that assigned (or corrected) tags based on the presence of certain cues. The first type of rules were crafted to correct any tag combinations that are 'impossible' according to the tagging scheme. For instance, when tagging a multi-word named entity the tags all have to be of the same type. This means that tag sequences such as `B-PER I-ORG` or `B-LOC I-LOC I-PER` are not

allowed. The problem with correcting these erroneous sequences lies in deciding which type to assign to the named entity. Several different options were investigated and in the case of the first kind of error (`B-PER I-ORG`) the best option turned out to be assigning the named entity the type of the first tag (`B-PER I-PER`). Erroneous tag sequences for named entities longer than two words (such as `B-LOC I-LOC I-PER`) were corrected by assigning every word of the named entity the majority type (`B-LOC I-LOC I-LOC`).

The second type of rules are similar to the example rules of section 2.1 where certain cues were used to assign named entity tags to words. For instance, a common error the classifier made was misclassifying certain locations as persons (`B-PER` instead of `B-LOC`). Many of these misclassifications were corrected using a rule that changed all `B-PER` tags to `B-LOC` if they were directly preceded by location cues such as "hoofdstad", "regio", "provincie", or "deelstaat". Appendix B contains all of the rules, successful or otherwise, that were created and tested in the post-processing stage.

The effectiveness of all of these rules was tested on the training and the development set[6]. The labels of the training set were predicted using the classifier stacking approach and 10-fold cross-validation. The following heuristic was used to determine if a rule was included in the final rulebase or not: if the sum of the changes on the training and the development set was positive, the rule was included. The improvements achieved by using the rulebase was less than the sum of its parts in that the sum of all individual improvements. However, this is natural because rule A might correct some of the errors that rule B would have corrected as well. We also experimented with ordering the rules differently, but this did not seem to affect the effectiveness of the rulebase at all. Even using the same rule twice at different stages of the process had no effect on the performance. All of the rules were implemented in AWK, one script for each rule. The post-processing experiments were conducted on a Pentium 4 2.0 GHz with 512 MB of RAM running SUSE 9.0.

[6]These rules were only tested on the training and development sets of the perfect chunk task and not on the other, best-performing dataset for lack of time.

Chapter 5

Experimental results

This chapter contains the results of all the experiments described in the previous chapter. The next section describes the feature subsets selected for the four different feature selection problems (`one-shot`, `ident`, `class.imperfect`, and `class.perfect`). The next four sections describe the outcome of the four rounds of experiments (basic, seedlists, stacking, and post-processing).

5.1 Feature selection

Table 5.1 contains the features selected by the SFFS algorithm. In addition to these features, all of the final feature subsets also contained the five words and the five part-of-speech tags and the relevant feature subsets also contained the named entity chunk tags.

	selected features
one-step classification	`firstCap_FOCUS`, `suffix_FOCUS`, `prefix_FOCUS`, `allLowercase_L1`, `containsPunctuation_FOCUS`, `suffix_L1`, `suffix_R1`, `prefix_L2`
chunking	`firstCap_FOCUS`, `firstCap_L1`, `prefix_FOCUS`, `firstCap_R1`, `suffix_L1`, `suffix_FOCUS`, `allLowercase_L2`
classification with imperfect chunks	`suffix_FOCUS`, `prefix_FOCUS`, `wordlength_FOCUS`, `prefix_L1`, `prefix_L2`
classification with perfect chunks	`suffix_FOCUS`, `containsPunctuation_FOCUS`, `prefix_FOCUS`, `prefix_R1`

Table 5.1: The suboptimal feature subsets for each of the four tasks. The features are listed in the order in which they were selected.

The results in table 5.1 show that the morphological information provided by the prefixes and suffixes of the windowed words was essential in predicting the named entity types. Each of the four subsets contained at least three morphological features; the prefix and suffix of the focus word were even included in every set. Prefixes and suffixes of the other words surrounding the focus word were also popular features, even though the exact position of the word/prefix in the window seemed to vary somewhat.

The feature set selected for one-step classification appeared to be largely a mix of the feature subsets selected for the NE chunking and classification tasks as it combined the most important chunking features with the most important classification features. The one-step subset also appeared to contain five morphological features, the biggest number in all four subsets.

The chunking subset contained the largest number of orthographic features (4) and these features were spread out over four of the five windowed words. Only orthographic information about the word two positions to the right of the focus word was not selected by SFFS. Apparently, determining the boundaries of a named entity is more dependent on orthographic changes (capitalization) than determining the type of a named entity is.

Two extra features were selected at some point in the process (`prefix_L2` and `wordlength_L1`), but they were removed at a later stage when the generalization performance could be increased by doing so. The `firstCap_FOCUS` feature seemed to be the most important feature for chunking and one-step classification because it was selected first by the SFFS algorithm for both tasks. The focus word suffix seems to be slightly more important for determining the type of NE than the focus word prefix, since it was selected earlier. In contrast, the prefix appears to be more important for the chunking task than the suffix.

Named entity classification seemed to depend mostly on morphological information: both the imperfect and perfect subsets contained only one non-morphological feature each. The difference between the make-up of the imperfect and perfect feature sets is striking: the difference between perfect and predicted chunks resulted in an overlap of only two features (`prefix_FOCUS` and `suffix_FOCUS`) and these two features were selected for each of the four tasks. Apparently, correcting the mistakes made in the chunking stage and determining the NE types requires different information than simply determining the NE types using perfect boundaries.

However, the `suffix_FOCUS` feature was the most important feature for both approaches, because it was selected first in both SFFS runs. The `containsPunctuation_FOCUS` feature was present in the one-step feature subset and in the subset for classification with perfect chunks. Apparently, punctuation information is more relevant for determining the type of a named entity than the boundaries of that named entity. The presence of the `wordlength_FOCUS` feature is peculiar since we did not expect it to be relevant for NE recognition.

5.2 Basic experiments

Table 5.2 contains the results of the basic experiments described in section 4.4. Both algorithms were used to classify both the development and the test set which yielded four scores for each task and a total of ten scores. Any decisions regarding the optimal approach were made using the results on the development set. The test scores will be described in greater detail here because they are independent and cannot suffer from overfitting like the development scores. This means that all scores are from the test set unless stated otherwise.

	TIMBL		MAXENT	
	development set	**test set**	**development set**	**test set**
one-step, selected features	58.38	60.03	54.75	57.40
one-step, all features	67.78	**70.58**	54.72	56.42
two-step, only chunking	87.31	88.58	84.41	82.86
two-step, imperfect chunking	54.52	57.44	60.61	61.17
two-step, perfect chunking	65.74	**70.07**	64.34	67.16

Table 5.2: F-scores of the basic experiments described in section 4.4. The best test scores for the one-step and two-step approach are printed in bold.

The results show that the k-NN algorithm outperformed the MAXENT algorithm in almost all of the experiments; only on the task of classification with imperfect chunks did MAXENT perform better. Overall, the best combination of features and algorithm in the basic experiments was using TIMBL with all of the 100 features which yielded an F-score of 70.58%. This was obviously also the best one-step score and in the one-step approaches TIMBL outperformed MAXENT by a difference of 3% or more in every experiment. The best score of a two-step approach was achieved by using TIMBL on the task of classification using perfect chunks (70.07%). In the two-step approaches, classification using perfect chunks outperformed classification using imperfect chunks for both TIMBL and MAXENT.

Feature selection appeared to be successful for MAXENT since all of the experiments involving feature subsets outperformed the experiments that used every feature by at least 1the one-step approaches, but large for the two-step approaches: with a score of 67.16% MAXENT performed almost 11% better on the task of classification with perfect chunks than it did using all features (the lowest MAXENT score at 56.42%).

However, the assumption made in section 4.2.2 appeared to be flawed: the feature set that was sub-optimal for the MAXENT algorithm did not turn out to be a good approximation of the sub-optimal feature set for the k-NN algorithm. The TIMBL scores were higher for almost every basic experiment and the supposed advantage of feature selection—less features can yield a higher generalization performance—was not true at all since the TIMBL score using all features was higher than the score of any of the experiments with 'specialized' feature subsets, often by a large margin of 10% to 13%. This is also where the biggest difference between the two algorithms can be found:

TIMBL performed 14% better using all features than MAXENT. Finally, TIMBL also outperformed MAXENT on the task of identifying the boundaries of named entities: the highest chunking score of 88.58% was achieved by TIMBL.

Both algorithms performed better on the test set than on the development set in 9 out of 10 experiments; only MAXENT performed better on the chunking task on the development set than on the test set. The highest score on the development set of 67.78% was also achieved by TIMBL using all features.

5.3 Seedlists

Table 5.3 contains the results of the experiments that incorporated seedlist features as described in section 4.5. Once again, both algorithms were used to classify both the development and the test set which resulted in four scores for each task and a total of ten scores.

	TIMBL		MAXENT	
	development set	**test set**	**development set**	**test set**
one-step, selected features	62.50	64.21	58.83	61.81
one-step, all features	67.93	**69.35**	55.63	58.20
two-step, only chunking	84.67	86.26	83.27	83.92
two step, imperfect chunking*	65.86	68.52	58.04	60.19
two step, perfect chunking	66.84	**70.13**	62.77	65.36

Table 5.3: The results of the experiments that used seedlist features as described in section 4.5. The best test scores for the one-step and two-step approach are printed in bold. (* = both of these MAXENT and TIMBL datasets used the chunk tags predicted in the first round of experiments, since these outperformed the chunking classifiers of this seedlist round)

The results show that TIMBL now outperformed the MAXENT algorithm in every experiment and that the difference between the performance of TIMBL and MAXENT increased even further. However, in this round the best score overall was achieved by the two-step approach with perfect chunking with an F-score of 70.13%. Because this was a ceiling performance, the best 'real' score was achieved by using TIMBL with all of the 100 features which yielded a slightly decreased F-score of 69.35%. This was also the best one-step score but not as good as the score in the previous round. In the one-step approaches TIMBL once again outperformed MAXENT by a difference of 3% or more in every experiment. In the two-step approaches, classification using perfect chunks still outperformed classification using imperfect chunks for both TIMBL and MAXENT with a bigger difference for MAXENT.

Using seedlist features improved the generalization performance for most of the experiments. It improved the F-scores on the one-step classification task for TIMBL and MAXENT with 4% to

64.21% and 61.81% respectively and the TIMBL score on the task of classification with imperfect chunks by 10% to 67.79%. Apparently, seedlist information was very useful for correcting chunking mistakes made by TIMBL.

The difference between the scores for classification with imperfect and perfect chunks decreased for both algorithms. This is especially evident for TIMBL where a gap of 13% was almost completely bridged with the score for classification with imperfect chunks going from 57.44% to 68.52%. Including seedlist information also resulted in a better performance of TIMBL on all of the two-step experiments, whereas MAXENT outperformed TIMBL on the task of classification with imperfect chunks in the previous round of experiments.

The combinations of algorithm and feature subset that performed best in the previous round of experiments hardly benefitted from the seedlist features. The score of MAXENT on the two-step approaches even dropped by about 2% and all of the scores of the chunking experiments did not benefit from the seedlist information at all as their scores were 1 to 2% lower. In general, adding seedlist information was bad for MAXENT performance on the two-step approaches as five of six scores actually decreased because of it. Finally, despite using the better chunks from the first round of experiments, MAXENT performed worse on the task of classification with perfect chunks.

Finally, after incorporating seedlist information both algorithms performed better on the test set than on the development set in all of the ten experiments.

5.4 Classifier stacking

Table 5.4 contains the results of the classifier stacking experiments as described in section 4.6. Classifier stacking was only performed for the two best-performing combinations of algorithm and feature subset in the previous rounds so this produced two scores for each task and a total of four scores. The one-step approach with all features performed the best in the previous two rounds so it was selected for this round in addition to the two-step approach with perfect chunking.

	TIMBL	
	development set	**test set**
one-step, all features	65.99	70.22
two step, perfect chunking	67.69	**71.65**

Table 5.4: The results of the classifier stacking experiments described in section 4.6.

The results show that again the ceiling performance of this two-step approach outperformed the one-step approach as TIMBL now achieved an F-score of 71.65% on the task of classification with perfect chunks whereas classification using all features increased again dropped by $\sim$1% to 70.22%. This means that classifier stacking certainly appeared to be helpful for both approaches even though it did not have a big influence: none of the scores were improved by more than

1.5%. However, feeding TIMBL more and more features did not appear to be as beneficial on the development set as it was on the test set. Adding the four extra stacking features on top of the 100 + 4 features was apparently too much for TIMBL in the development set.

Appendix C contains the suboptimal parameter settings found using the PARAMSEARCH program for this round of experiments and the previous two rounds. The most popular feature weighting metric was gain ratio and the inverse linear weighting scheme was selected the most as the voting scheme. The number of nearest neighbors ranged from 1 to 25. The Jefferson divergence metric was selected most often as the number of features in a dataset increased which is in keeping with its robustness when it comes to sparse data.

5.5 Post-processing

Table 5.5 contains the results of the post-processing experiments described in section 4.7. Overall the best test score was achieved by TIMBL in the first round using just the 100 initial features: 70.58%. The results of this classifier were used in the post-processing experiments, along with the results of the two-step perfect chunking approach (71.65%).

	TIMBL	
	development set	**test set**
one-step, all features	69.26	71.90
two step, perfect chunking	72.74	76.30

Table 5.5: The results of the post-processing experiments described in section 4.7. Table 5.6 contains an overview of the effectiveness of each separate handcrafted rule.

The results in table 5.5 show that the handcrafted rules used to correct the most frequent errors were successful: improvements ranged from 1.7% to 4.7%. The best score of the one-step approach using all features was improved to 71.90% which is the sixth best performance of all submissions in the CoNLL 2002 shared task [Tjong Kim Sang, 2002a]. It also outperformed all other submitted approaches that used memory-based learners. The ceiling performance came in second place which shows the potential for two-step approaches.

Table 5.6 shows the effectiveness of the each of the separate rules used in the post-processing experiments. Appendix B contains each of the twelve rules that were created and tested in the post-processing stage.

Four of the twelve rules had a negative effect on the generalization performance; two of those rules did improve performance on the development set but not on the training set. Rule 2 is an good example of this: it decreased performance by ∼6% on the training set and only increased it by ∼0.5% on the development set. Rules 7 and 8 are mutually exclusive (as explained in appendix B) and rule 8 was included in the final set of rules because it outperformed rule 7 by 4.39% to 3.65%. Another well-performing rule, also of type I, was rule 1 which improved accuracy by 1.67% when applied individually. In general, I-type rules performed much better than rules of

rule no.	training set	development set	sum	type
1	0.1178%	1.5551%	1.6729%	I
2	–5.9715%	0.5497%	–5.4218%	C
3	–2.1732%	–0.6969%	–2.8701%	C
4	–0.0185%	0.2683%	0.2498%	C
5	–0.0075%	0.1223%	0.1148%	C
6	–0.0185%	0.0366%	0.0181%	C
7	0.2870%	3.3662%	3.6532%	I
8	0.1785%	4.2134%	4.3919%	I
9	0.0150%	0.3423%	0.3573%	C
10	–0.0335%	0.1087%	0.0752%	C
11	–1.0570%	–0.9168%	–1.9738%	C
12	–0.1216%	0.0490%	–0.0726%	C

Table 5.6: The effectiveness of the 12 separate post-processing rules. The fourth column shows the type of rule: rules that corrected impossible tag combinations (I) or rules that assigned (or corrected) tags based on the presence of certain cues (C).

type C. The more 'speculative' of the C-type rules (2, 11, and 12) did not perform well whereas the simple pattern-matching rules (4, 5, 6, 9, and 10) did result in small performance gains. Of these five rules, rule 9 was the best with a combined performance gain of 0.36%.

Chapter 6

Discussion & future research

In this chapter we return to the research questions posed in chapter 1 and determine if these have been answered sufficiently. Answering the first research question consisted of two parts: determining the optimal set of features and determining the best algorithm and algorithm parameters. Both steps can improve generalization performance but are not independent from one another as was argued in section 3.3. Since joint optimization of these two tasks would be computationally very expensive, we opted for a compromise between independent feature selection and parameter optimization and an interleaved solution. We performed a virtually completely interleaved feature selection and parameter optimization process using an algorithm with very few parameters (MAXENT). We proceeded by assuming that the same feature set that was sub-optimal for this algorithm would be a good approximation of the sub-optimal feature set for the second machine learning algorithm (k-NEAREST NEIGHBOR).

Because of the second research question, we examined two different approaches of NER (splitting it up into separate stages or predicting it in one big step). This meant we had to repeat the feature selection process described above several times resulting in four different feature subsets. Several conclusions can be drawn from these feature selection experiments.

6.1 Features

Morphological features were found to be very important for recognizing named entities: each of the four subsets contained at least three morphological features and the prefix and suffix of the focus word were included in every set. This kind of information is important because similar prefixes and suffixes can often be found in groups of named entities of the same type. For instance, prefixes and suffixes such as “San” and “-aard” often signal locations (“San Sebastian”, “San Simeon”, “Dedemswaard”, or “Heerhugowaard”).

Morphological features appear to have the biggest influence on the tasks where the complete NE tags (boundary and type) have to be predicted. Both of the tasks of the two-step approaches of classification and the one-step approach rely heavily on morphological features. The suffix of the focus word is especially important since it was selected first or second for these three feature sets. The one-step approach depended the most on morphological information: this is because in the one-step approach nine-valued tags have to be predicted in one step instead of three-valued tags in one step or nine-valued tags in two steps. To a lesser extent, the morphological features are also relevant for the identification task.

Orthographic features (and in particular capitalization features) are the most important for tasks where the boundaries of named entities have to be determined. Features that represent the presence or absence of word-initial capitalization (`firstCap` and `allLowercase`) occupy the most prominent place in the set of orthographic features. This is in agreement with the intuitive rule mentioned in chapter 1 that named entities are often proper names and have initial capital letters. The `firstCap_FOCUS` feature is the most important feature because it was selected first by SFFS for all chunking tasks. Another orthographic feature that has predictive value is `containsPunctuation_FOCUS`; however, this appears to be more relevant for classification of the NE type than determining the boundaries.

The absence of certain features such as `firstSentenceWord` and `functionWord` from all of the selected feature sets is peculiar: they are often used in the handcrafted approaches described in 2.1 to filter out function words in sentence-initial position. Apparently, morphological features have more predictive power for an algorithm such as MAXENT.

The task of one-step recognition required the largest number of features of all four tasks which is probably due to the fact that it has to perform two tasks at the same time: identifying the boundaries and determining the type of a named entity. Each of these tasks has different characteristics and requires different kinds of information and this explains why the selected feature set was largely a mix of the most important chunking features and the most important classification features.

The substantial difference between the make-up of the imperfect and perfect feature sets (an overlap of only two features) shows that correcting the mistakes made in the chunking stage and determining the NE types requires different information than simply determining the NE types using perfect boundaries. The presence of the `wordlength_FOCUS` feature in the 'imperfect chunks'-set is peculiar and we have no explanation for its relevance to NER.

Seedlist features are also successful in improving the generalization performance, even though the actual influence is highly dependent on the classification algorithm. TIMBL seemed to benefit more from the inclusion of seedlists than MAXENT, where the performance of the majority of tasks actually decreased after inclusion. Seedlist features have the greatest effect on tasks where the type of a NE has to be determined: especially the one-step approach and classification using imperfect chunks benefitted from the use of gazetteers.

Seedlist features do not help with identifying the boundaries of named entities: generalization

performance on the NE chunking task decreased after gazetteer inclusion. A possible explanation for this is that orthographic features are simply the most relevant features for chunking; adding the four seedlist features can only serve to confuse the classifier.

Using classifier stacking to add the predictions of another level classifier as features also appears to improve generalization performance. Stacking was not used for every task in third round of experiments but most of the experiments benefitted from the new windowed features. However, the actual improvement depends a great deal on the classification algorithm: adding the features to an already large feature set can actually decrease performance by the k-NN algorithm. We will discuss this further in the next section. At any rate, the value of classifier stacking is certainly reinforced by the available literature.

Overall, seedlist and stacking features have a positive influence despite not being included in the best-performing approach: the one-step approach of using all available features except seedlist information. The feasture set for classification with imperfect chunks also performed well.

6.2 Algorithms

The basic experiments confirmed the advantages of feature selection mentioned in chapter 3: it was successful in improving the generalization performance for the algorithm used in the process. The MAXENT algorithm performed better in all of the experiments involving feature subsets than in the experiments that used every feature.

Our assumption that the sub-optimal feature set for the MAXENT algorithm would be a good approximation of the sub-optimal feature set for the k-NN algorithm turned out to be unjustified. Feature selection is dependent on the algorithm used in the evaluation function and therefore our expectations were that TIMBL would perform a little worse on the same feature sets. However, the assumptions and expectations were flawed since the TIMBL scores were higher for almost every basic experiment. Furthermore, the set of all features outperformed all of the 'specialized' feature sets selected using SFFS. A possible explanation for these peculiar results could be that the k-NN algorithm is better suited for NER than MAXENT algorithm. However, this is not in agreement with the findings in the literature where MAXENT is one of the most popular approaches to NER. A better explanation is that certain features are more important for MAXENT while other features are better for TIMBL and these two optimal feature sets do not overlap completely. It is reasonable to assume that not all of the features optimal for TIMBL were included in the feature sets selected for MAXENT. However, these features were obviously present in the original 100-feature dataset. Apparently, these 'missing' features are important enough for TIMBL to perform better on the complete dataset.

The best performance by both algorithms was TIMBL's on the one-step approach with all features. It also performed well on the two-step task with perfect chunking. This task dramatically outperformed classification using imperfect chunks for both TIMBL and MAXENT in the basic experiments. This is perfectly logical, because the ceiling performance of the perfect chunk classification does not have to correct for any additional errors in the identification of the NE boundaries.

The advantage of TIMBL over MAXENT increased even more with the addition of seedlist features as TIMBL outperformed MAXENT by 3% or more. After adding these features the best performance was still TIMBL's one-step approach with all the features but the differences between the different TIMBL performances decreased. The task of classification with imperfect chunks benefits the most from the seedlist features as its performance score increased by ∼11%. Apparently, seedlist information is very useful for correcting chunking mistakes made by TIMBL.

However, this makes it peculiar that seedlist features do not help at all in identifying the boundaries of named entities as the scores of both algorithms decreased on the chunking task. A possible explanation for this, also mentioned in the previous section, is that orthographic features are simply the most relevant features for chunking; adding the four seedlist features can only serve to confuse the classifier.

The effects of classifier stacking on the performance of MAXENT were not examined but adding the windowed predictions of the first level classifier to the datasets improved generalization performance of TIMBL. Even though stacking was not used for every task in third round of experiments, the task of classification with perfect chunks benefitted from the new windowed features.

However, the actual improvement seems to depend on both the feature set and the classification algorithm as classifier stacking actually decreased the performance of TIMBL on the set of all features. In general, supplying TIMBL with more and more features did not appear to be beneficial: adding the four seedlist features in the previous round yielded the smallest improvement and adding the four extra stacking features on top of the 100 + 4 features actually decreased performance. Part of the explanation for this was already given in the previous section: the features best suited for TIMBL were not selected for MAXENT in the feature selection process which is a very likely reason for TIMBL's better performance on the set containing all features. The most likely reason that adding the stacking features actually decreased generalization performance is a disadvantage shared by most nearest neighbor approaches: they typically consider all attributes of the instances when determining the nearest neighbors. This means that irrelevant features can have a negative impact on the distance metric as the distance between neighbors will be completely dominated a large number of irrelevant attributes [Mitchell, 1997]. The stacking attributes need not be irrelevant for the task but it is possible that they shift this delicate balance in the wrong direction.

Handcrafted rules are a very effective way of improving upon the generalization performance of a machine learning approach by correcting the most frequent errors made by the classifier(s). The rules used in the post-processing stage improved the F-scores achieved by TIMBL by at least 1.7%. The usefulness of such a hybrid approach is further supported by the literature: the best-performing system at the MUC-7 conference was the hybrid approach of [Mikheev et al., 1999] and handcrafted filter rules were also used by [Tjong Kim Sang, 2003] to correct classifier errors.

The I-type rules that corrected impossible tag combinations performed better than the C-type rules that assigned (or corrected) tags based on the presence of certain cues. A plausible explanation for this is that the I-type rules apply to all four types of NE tags (`LOC`, `ORG`, `MISC`, and `PER`) whereas the C-type rules are much more specific, often only changing one type of tag to another.

6.3 Conclusions

Overall, the following conclusions can be drawn from the experiments with regard to the research questions:

- The best approach to NER is using all of the available feature and classifying the named entities in one step. However, the ceiling performance of the two-step approach suggests that splitting the recognition process into separate identification and classification phases can be even more beneficial, especially if feature selection is used more extensively for the k-NN algorithm. If we disregard the best approach using all features, then both of the two-step approaches outperformed the specialized one-step approach in every experiment round for almost every classification algorithm. The MAXENT also seem to corroborate this: after feature selection the two-step approaches outperformed the one-step approaches.

- Morphological features (prefixes and suffixes) are good indicators of named entity type, especially in tasks where, among other things, the category of a named entity (`LOC`, `ORG`, and so on) has to be predicted, but also when the boundaries of a named entity need to be predicted. Orthographic features that represent the capitalization characteristics of a word are especially useful in predicting these boundaries. Seedlist features are also very helpful in NER, but the actual improvement is highly dependent on the classification algorithm used. Like morphological features, seedlist information has the greatest effect on tasks where the type of a NE has to be determined. Boundary identification does not benefit from the use of gazetteers. Using the stacked predictions of another level classifier as features also improves generalization performance.
  The feature set that gave the best performance after the experiment rounds contained all of the 100 initial features. The suboptimal feature set of the two-step ceiling performance contained the following 22 features: five windowed words and their POS tags, `suffix_FOCUS`, `containsPunctuation_FOCUS`, `prefix_FOCUS`, `prefix_R1`, four seedlist features, and four features representing the windowed predictions of the first level classifier.

- The experiments confirmed the prevalent views that feature selection is highly dependent on the classification algorithm: selected feature subsets are not easily transferable.

- The k-NEAREST NEIGHBOR algorithm outperformed the MAXENT algorithm in most of the experiments as the latter algorithm did not live up to its reputation as one of the algorithms best suited to the task of NER. The following settings gave the best results on the best-performing task (one-step classification using all 100 basic features): the k-NN algorithm

with 7 nearest neighbors used for extrapolation, the Jefferson divergence metric with the gain ratio measure as feature weighting and inverse linear weighting as the voting scheme.

- Using handcrafted rules to correct frequent errors made by the classifier is a successful technique for boosting the generalization performance on the task of NER.

6.4 Future research

Several recommendations can be made for future research into NER and one of the obvious recommendations is examining the usefulness of even more features. We sampled the NER literature for useful features and used the 20 most popular features, but there are more features in the literature that might have predictive value. Adding semantic tags or features that encode whether or not a word already occurred in lowercase or uppercase form in the current article/document (similar to the approach of [Mikheev et al., 1999]) might help to improve generalization performance. The performance of the morphological features might be improved as well by using a specialized database such as Celex not only to extract more accurate prefixes and suffixes but perhaps to include inflection and lemmatization information. Different kinds of seedlist features might also be helpful such as features that encode whether a word can be the first part of a NE or not.

We added the seedlist features in a separate stage to better measure their influence on the generalization performance but it is also possible to add the seedlist features before the feature selection stage. This way, if a certain seedlist feature is not helpful in predicting, for instance, the NE boundaries, then it need not have a negative effect on performance as the won't be selected in the feature selection stage. This inclusion before feature selection goes for almost every feature: it might be wise to only give the focus word itself a fixed place in the feature subsets and let the feature selection algorithm determine whether including the other windowed words or POS tags is beneficial to the performance. Increasing window size from 2-1-2 to 3-1-3 or 4-1-4 might also be useful despite the timeconsuming nature of using larger windows and thus more features in the feature selection process.

Another issue of possible improvement would be to interleave the feature selection and parameter optimization process for TIMBL. This is likely to yield improvement since the performance of TIMBL on the set containing all features showed that the MAXENT feature sets did not contain all of the features optimal for TIMBL. In addition, the ceiling performance of the two-step approach suggests that splitting the recognition process can be even more beneficial. Better feature selection for TIMBL can be a means of fulfilling this promise. However, completely interleaving feature selection and parameter optimization is computationally very expensive because of TIMBL's many parameters. One successful solution would be to use genetic algorithms to handle these large search spaces, despite their computationally expensive nature.

The potential of splitting up the NER recognition process into two separate stages makes it interesting to investigate whether splitting these tasks even further is an even more successful way of addressing the problem of NER. Combining the results of several binary classifiers, one for each category of named entity and one for each type of boundary tags, might work better on the

same grounds that the two-step approach outperformed the one-step approach: the optimal feature sets for the identification and classification stages are different as determining the boundaries of named entities requires different information than determining its class. It is possible that classifying a LOC named entity requires different features and parameter settings than classifying a PER named entity; these suspicions seem to be confirmed by the findings of [Carreras et al., 2003]. A good approach for combining the results of these binary classifiers would be boosting, which is also a very popular technique for NER. Finally, despite our reasons for selecting the current approach to classifier stacking, perhaps another approach than the one described in section 4.6 might yield improvements as well.

Another interesting area of improvements in performance on the NER task is using error-correcting rules. The time spent on the post-processing stage was the shortest of all experiments; therefore, a more extensive investigation of the use of handcrafted rules in a hybrid approach seems promising.

Appendix A

Example instances from a windowed dataset

```
communicatiebureau dat Floralux inhuurde .  N Conj N V Punc firstCap_NO
firstCap_NO firstCap_YES firstCap_NO firstCap_NO allCaps_NO allCaps_NO allCaps_NO
allCaps_NO allCaps_NO allLowercase_YES allLowercase_YES allLowercase_NO
allLowercase_YES allLowercase_NO internalCaps_NO internalCaps_NO internalCaps_NO
internalCaps_NO internalCaps_NO containsDigit_NO containsDigit_NO containsDigit_NO
containsDigit_NO containsDigit_NO containsDigitAndAlpha_NO containsDigitAndAlpha_NO
containsDigitAndAlpha_NO containsDigitAndAlpha_NO containsDigitAndAlpha_NO onlyDigits_NO
onlyDigits_NO onlyDigits_NO onlyDigits_NO onlyDigits_NO isPunctuation_NO isPunctuation_NO
isPunctuation_NO isPunctuation_NO isPunctuation_YES containsPunctuation_NO
containsPunctuation_NO containsPunctuation_NO containsPunctuation_NO
containsPunctuation_YES isHyphenated_NO isHyphenated_NO isHyphenated_NO isHyphenated_NO
isHyphenated_NO 0 0 0 0 1 firstSentenceWord_NO firstSentenceWord_NO firstSentenceWord_NO
firstSentenceWord_NO firstSentenceWord_NO quotedText_NO quotedText_NO quotedText_NO
quotedText_NO quotedText_NO isURL_NO isURL_NO isURL_NO isURL_NO isURL_NO 18 3 8 8 1
functionWord_NO functionWord_YES functionWord_NO functionWord_NO functionWord_NO comm
dat Flor inhu - reau dat alux urde - B-ORG

dat Floralux inhuurde .  -EMPTY- Conj N V Punc -EMPTY- firstCap_NO firstCap_YES
firstCap_NO firstCap_NO firstCap_NO allCaps_NO allCaps_NO allCaps_NO allCaps_NO
allCaps_YES allLowercase_YES allLowercase_NO allLowercase_YES allLowercase_NO
allLowercase_NO internalCaps_NO internalCaps_NO internalCaps_NO internalCaps_NO
internalCaps_YES containsDigit_NO containsDigit_NO containsDigit_NO
containsDigit_NO containsDigit_NO containsDigitAndAlpha_NO containsDigitAndAlpha_NO
containsDigitAndAlpha_NO containsDigitAndAlpha_NO containsDigitAndAlpha_NO onlyDigits_NO
onlyDigits_NO onlyDigits_NO onlyDigits_NO onlyDigits_NO isPunctuation_NO
```

isPunctuation_NO isPunctuation_NO isPunctuation_YES isPunctuation_NO
containsPunctuation_NO containsPunctuation_NO containsPunctuation_NO
containsPunctuation_YES containsPunctuation_YES isHyphenated_NO isHyphenated_NO
isHyphenated_NO isHyphenated_NO isHyphenated_NO 0 0 0 1 0 firstSentenceWord_NO
firstSentenceWord_NO firstSentenceWord_NO firstSentenceWord_NO firstSentenceWord_NO
quotedText_NO quotedText_NO quotedText_NO quotedText_NO quotedText_NO isURL_NO isURL_NO
isURL_NO isURL_NO isURL_NO 3 8 8 1 7 functionWord_YES functionWord_NO functionWord_NO
functionWord_NO functionWord_NO dat Flor inhu - - dat alux urde - - O

Floralux inhuurde . -EMPTY- -EMPTY- N V Punc -EMPTY- -EMPTY- firstCap_YES
firstCap_NO firstCap_NO firstCap_NO firstCap_NO allCaps_NO allCaps_NO allCaps_NO
allCaps_YES allCaps_YES allLowercase_NO allLowercase_YES allLowercase_NO
allLowercase_NO allLowercase_NO internalCaps_NO internalCaps_NO internalCaps_NO
internalCaps_YES internalCaps_YES containsDigit_NO containsDigit_NO containsDigit_NO
containsDigit_NO containsDigit_NO containsDigitAndAlpha_NO containsDigitAndAlpha_NO
containsDigitAndAlpha_NO containsDigitAndAlpha_NO containsDigitAndAlpha_NO onlyDigits_NO
onlyDigits_NO onlyDigits_NO onlyDigits_NO onlyDigits_NO isPunctuation_NO isPunctuation_NO
isPunctuation_YES isPunctuation_NO isPunctuation_NO containsPunctuation_NO
containsPunctuation_NO containsPunctuation_YES containsPunctuation_YES
containsPunctuation_YES isHyphenated_NO isHyphenated_NO isHyphenated_NO isHyphenated_NO
isHyphenated_NO 0 0 1 0 0 firstSentenceWord_NO firstSentenceWord_NO firstSentenceWord_NO
firstSentenceWord_NO firstSentenceWord_NO quotedText_NO quotedText_NO quotedText_NO
quotedText_NO quotedText_NO isURL_NO isURL_NO isURL_NO isURL_NO isURL_NO 8 8 1 7 7
functionWord_NO functionWord_NO functionWord_NO functionWord_NO functionWord_NO Flor
inhu - - - alux urde - - - O

-EMPTY- -EMPTY- In ’81 regulariseert -EMPTY- -EMPTY- Prep Num V firstCap_NO
firstCap_NO firstCap_YES firstCap_NO firstCap_NO allCaps_YES allCaps_YES
allCaps_NO allCaps_NO allCaps_NO allLowercase_NO allLowercase_NO allLowercase_NO
allLowercase_NO allLowercase_YES internalCaps_YES internalCaps_YES internalCaps_NO
internalCaps_NO internalCaps_NO containsDigit_NO containsDigit_NO containsDigit_NO
containsDigit_YES containsDigit_NO containsDigitAndAlpha_NO containsDigitAndAlpha_NO
containsDigitAndAlpha_NO containsDigitAndAlpha_NO containsDigitAndAlpha_NO onlyDigits_NO
onlyDigits_NO onlyDigits_NO onlyDigits_NO onlyDigits_NO isPunctuation_NO isPunctuation_NO
isPunctuation_NO isPunctuation_NO isPunctuation_NO containsPunctuation_YES
containsPunctuation_YES containsPunctuation_NO containsPunctuation_YES
containsPunctuation_NO isHyphenated_NO isHyphenated_NO isHyphenated_NO isHyphenated_NO
isHyphenated_NO 0 0 0 0 0 firstSentenceWord_NO firstSentenceWord_NO firstSentenceWord_NO
firstSentenceWord_NO firstSentenceWord_NO quotedText_NO quotedText_NO quotedText_NO
quotedText_NO quotedText_NO isURL_NO isURL_NO isURL_NO isURL_NO isURL_NO 7 7 2 3 13
functionWord_NO functionWord_NO functionWord_YES functionWord_NO functionWord_NO - -
In 81 regu - - In 81 eert O

Appendix B

Handcrafted post-processing rules

It should be noted that no rule was created to prevent impossible sequences such as "`... O I-LOC ...`" because they did not occur in the predictions of the classifier after classifier stacking approaches. Apparently, the classifier learned not to predict these impossible sequences.

Rule 1

Type: Correct impossible tag sequences (I)

Description: Repair impossible chains of 3 consecutive NE tags if the types do not match by assigning the majority type to all three tags. This means that tag sequences such as `B-X I-Y I-X` or `I-X I-X I-Y` were changed to `B-X I-X I-X` and `I-X I-X I-X` respectively.

Rule 2

Type: Correct errors using phrasal and word cues (C)

Description: Words between brackets that are tagged as locations are usually organizations. This means that in word sequences such as `( Microsoft )` words that were erroneously tagged as `B-LOC` were changed to `B-ORG`.

Rule 3

Type: C

Description: Words classified as locations with a preceding determiner are often an organization. The `B-LOC` tag was changed to `B-ORG`.

Rule 4

Type: C

Description: If certain organizational cues occur before a word tagged as a location or a person, then it's often an organization. The following cues triggered a change from `B-LOC` or `B-PER` to `B-ORG`: *bedrijf*, *universiteit*, *rijksuniversiteit*, *organisatie*, *werkgroep*, *stichting*, *uitgever*, *krant*, and *concern*.

Rule 5

Type: C

Description: If the words *minister van* occur before a word tagged as location or miscellaneous, then it is often an organization. For instance, in the word sequence `minister van Defensie` the word `Defensie` should be tagged as `B-ORG`[1].

Rule 6

Type: C

Description: If the certain cues occurs two positions before the focus word tagged as a location, then it's often an organization. The following cues triggered a change from `B-LOC` to `B-ORG`: *commissie* and *centrum*. For instance, in the word sequence `Centrum voor Levensvragen` the type of the word `Defensie` should be changed to `ORG`.

Rule 7

Type: I

Description: Repair impossible chains of 2 consecutive NE tags if the types do not match. Assign the type of the last tag to both tags. This means that tag sequences such as `B-X I-Y` were changed to `B-Y I-Y`. This rule is the exact opposite of rule 8 which means that, in case they are both successful, only the best performing rule should be applied: using one of them would be make the other obsolete.

Rule 8

Type: I

Description: Repair impossible chains of 2 consecutive NE tags if the types do not match. Assign the type of the first tag to both tags. This means that tag sequences such as `B-X I-Y` were changed to `B-X I-X`. This rule is the exact opposite of rule 7 which means that, in case they are both successful, only the best performing rule should be applied: using one of them would be make the other obsolete.

Rule 9

Type: C

Description: If honorifics or titles precede words that were tagged as locations, then they should be changed to a person. The following cues triggered a change from `B-LOC` to `B-PER`: *minister*, *professor*, *president*, *premier*, *ex-president*, *dokter*, *dr.*, *eerwaarde*, and *meester*.

Rule 10

Type: C

Description: Certain locations are often mistaken for person names. If certain location cues are present before words tagged as a person, then they should be tagged as a location. The following cues triggered a change from `B-PER` to `B-LOC`: *in*, *uit*, *provincie*, *deelstaat*, *hoofdstad*, and *regio*.

[1] This is according to the CoNLL rules; another option would be to tag the entire sequence as a person.

Rule 11

Type: C

Description: If a word tagged as a person is preceded by a geographical adjective (such as *Nederlandse*), then they should be tagged as location instead of a person. Geographical adjectives are tagged as `B-MISC`, so sequences of `B-MISC B-PER` were changed to `B-MISC B-LOC`.

Rule 12

Type: C

Description: If another single word NE occurs two positions left of the focus word and is in disjunction[2] with the focus word, then the focus word should be assigned the same type. This means that sequences such as "`Nederland of Spanje`" that were tagged as `B-LOC O B-MISC` were changed to `B-LOC O B-LOC`.

[2]Experiments showed that testing for the presence of conjunctions did not have any corrective power.

Appendix C

Optimal Timbl settings

	Basic experiments			
	k	**distance metric**	**feature weighting**	**voting scheme**
one-step, selected features	1	MVDM	info gain	majority voting
one-step, all features	7	Jefferson	gain ratio	inverse linear
two-step, only chunking	5	overlap	gain ratio	inverse linear
two-step, imperfect chunking	11	overlap	gain ratio	majority voting
two-step, perfect chunking	5	MVDM	gain ratio	inverse linear

Table C.1: The optimal settings of TIMBL in the round of basic classification experiments; all of the settings had IB1 as the classification algorithm.

	Seedlist experiments			
	k	**distance metric**	**feature weighting**	**voting scheme**
one-step, selected features	3	Jefferson	no weighting	inverse distance
one-step, all features	7	Jefferson	gain ratio	inverse linear
two-step, only chunking	3	overlap	gain ratio	inverse linear
two-step, imperfect chunking	19	MVDM	gain ratio	inverse linear
two-step, perfect chunking	25	Jefferson	gain ratio	inverse distance

Table C.2: The optimal settings of TIMBL in the round of experiments that incorporated seedlist features; all of the settings had IB1 as the classification algorithm.

	Stacking experiments			
	k	**distance metric**	**feature weighting**	**voting scheme**
one-step, all features	7	Jefferson	gain ratio	inverse linear
two-step, perfect chunking	7	overlap	gain ratio	inverse linear

Table C.3: The optimal settings of TIMBL in the round of classifier stacking experiments; all of the settings had IB1 as the classification algorithm.

Bibliography

[Bagola, 2003] Bagola, H. (2003). Informations utiles à l'intégration de nouvelles langues européennes. Technical Report 1.10, Office des publications EU, DIR/A-Cellule. `http://publications.eu.int/images/pdf/fr/elarg-vl.pdf`.

[Bender et al., 2003] Bender, O., Och, F. J., and Ney, H. (2003). Maximum Entropy Models for Named Entity Recognition. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL-2003*, pages 148–151. Edmonton, Canada.

[Berger et al., 1996] Berger, A. L., Pietra, S. D., and Pietra, V. J. D. (1996). A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1):39–71.

[Bikel et al., 1997] Bikel, D. M., Miller, S., Schwartz, R., and Weischedel, R. (1997). Nymble: a high-performance learning name-finder. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, pages 159–168.

[Black et al., 1998] Black, W., Rinaldi, F., and Mowatt, D. (1998). FACILE: Description of the NE system used for MUC. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*. Morgan Kaufmann.

[Borthwick, 1999] Borthwick, A. (1999). *A Maximum Entropy Approach to Named Entity Recognition*. PhD thesis, New York University.

[Borthwick et al., 1998] Borthwick, A., Sterling, J., Agichtein, E., and Grishman, R. (1998). NYU: Description of the MENE named entity system as used in MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*. Morgan Kaufmann.

[Brill, 1995] Brill, E. (1995). Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*.

[Buchholz, 2002] Buchholz, S. (2002). *Memory-Based Grammatical Relation Finding*. PhD thesis, Tilburg University.

[Buchholz and Van den Bosch, 2000] Buchholz, S. and Van den Bosch, A. (2000). Integrating seed names and n-grams for a named entity list and classifier. In *Proceedings of LREC-2000*, pages 1215–1221, Athens, Greece.

[Carreras et al., 2002] Carreras, X., Màrques, L., and Padró, L. (2002). Named entity extraction using AdaBoost. In *Proceedings of CoNLL-2002*, pages 167–170. Taipei, Taiwan.

[Carreras et al., 2003] Carreras, X., Màrquez, L., and Padró, L. (2003). Learning a perceptron-based named entity chunker via online recognition feedback. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL-2003*, pages 156–159. Edmonton, Canada.

[Church and Mercer, 1993] Church, K. and Mercer, R. (1993). Introduction to the special issue on computational linguistics using large corpora. *Computational Linguistics*, 19(1):1 – 24.

[Collins, 2002] Collins, M. (2002). Ranking algorithms for named entity extraction: Boosting and the voted perceptron. In *Proceedings of ACL'02*.

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

[Cucchiarelli and Velardi, 2001] Cucchiarelli, A. and Velardi, P. (2001). Unsupervised named entity recognition using syntactic and semantic contextual evidence. *Computational Linguistics*, 27(1):123–131.

[Cucerzan and Yarowsky, 1999] Cucerzan, S. and Yarowsky, D. (1999). Language independent named entity recognition combining morphological and contextual evidence. In *Proceedings 1999 Joint SIGDAT Conference on EMNLP and VLC*.

[Daelemans et al., 2003a] Daelemans, W., Hoste, V., De Meulder, F., and Naudts, B. (2003a). Combined Optimization of Feature Selection and Algorithm Parameters in Machine Learning of Language. In Lavrac, N., Gamberger, D., Todorovski, L., and Blockeel, H., editors, *Proceedings of ECML 2003*, pages 84–95, Cavtat-Dubrovnik, Croatia. Berlin, Springer.

[Daelemans et al., 2003b] Daelemans, W., Zavrel, J., Van der Sloot, K., and Van den Bosch, A. (2003b). MBT: Memory-Based Tagger version 2.0. Reference guide 03-13, ILK. `http://ilk.uvt.nl/software.html#mbt`.

[Daelemans et al., 2003c] Daelemans, W., Zavrel, J., van der Sloot, K., and Van den Bosch, A. (2003c). TiMBL: Tilburg Memory Based Learner, version 5.0, Reference Guide. Reference guide 03-10, ILK. `http://ilk.uvt.nl/software.html#timbl`.

[Dake, 2003] Dake, J. (2003). Explorations of the speed-accuracy trade-off in memory-based learning algorithms. Technical Report ILK-0302, ILK Research Group. `http://ilk.uvt.nl/papers.html`.

[Dash and Liu, 1997] Dash, M. and Liu, H. (1997). Feature selection for classification. *Intelligent Data Analysis*, 1(3):131–156.

[De Meulder and Daelemans, 2003] De Meulder, F. and Daelemans, W. (2003). Memory-based named entity recognition using unannotated data. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL-2003*, pages 208–211. Edmonton, Canada.

[ESAT/PSI speech group, 2004] ESAT/PSI speech group (2004). The Atranos Project. http://atranos.esat.kuleuven.ac.be/.

[Ferri et al., 1994] Ferri, F., Pudil, P., Hatef, M., and Kittler, J. (1994). Comparative study of techniques for large-scale feature selection. In Gelsema, E. S. and Kanal, L. N., editors, *Pattern Recognition in Practice IV*, pages 403–413. Elsevier.

[Florian, 2002] Florian, R. (2002). Named entity recognition as a house of cards: Classifier stacking. In *Proceedings of CoNLL-2002*, pages 175–178. Taipei, Taiwan.

[Grishman, 1995] Grishman, R. (1995). The NYU system for MUC-6 or where's the syntax? In *Proceedings of the Sixth Message Understanding Conference*. Morgan Kaufmann.

[Hammerton, 2003] Hammerton, J. (2003). Named entity recognition with long short-term memory. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL-2003*, pages 172–175. Edmonton, Canada.

[Hearst et al., 1998] Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., and Schölkopf, B. (1998). Support vector machines. *IEEE Intelligent Systems*, ?(?):18–28.

[Hendrickx and Van den Bosch, 2003] Hendrickx, I. and Van den Bosch, A. (2003). Memory-based one-step named-entity recognition: Effects of seed list features, classifier stacking, and unannotated data. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL-2003*, pages 176–179. Edmonton, Canada.

[Hsu, 2001] Hsu, W. H. (2001). *Combining Classifiers: Weighted Majority, Bagging, and Stacking*. Lecture 21, Fall 2001, http://www.cis.ksu.edu/~bhsu, Department of Computing and Information Sciences, Kansas State University. Last visited: May 3rd, 2004.

[ILK, 2004] ILK (2004). Induction of Linguistic Knowledge: Machine Learning for Language Engineering and Linguistics. http://ilk.uvt.nl/.

[Jackson and Moulinier, 2002] Jackson, P. and Moulinier, I. (2002). *Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorization*, volume 5 of *Natural Language Processing*. John Benjamins Publishing Co., Amsterdam, 1st edition.

[Jain et al., 2000] Jain, A., Duin, R., and Mao, J. (2000). Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37.

[Jain and Zongker, 1997] Jain, A. and Zongker, D. (1997). Feature selection: evaluation, application, and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):153–158.

[Jansche, 2002] Jansche, M. (2002). Named entity extraction with conditional markov models and classifiers. In *Proceedings of CoNLL-2002*, pages 179–182. Taipei, Taiwan.

[Klein et al., 2003] Klein, D., Smarr, J., Nguyen, H., and Manning, C. D. (2003). Named entity recognition with character-level models. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL-2003*, pages 180–183. Edmonton, Canada.

[Kohavi, 1995] Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*.

[Krupka and Hausman, 1998] Krupka, G. R. and Hausman, K. (1998). Description of the NetOwl Extractor System as used for MUC-7. In *Proceedings of the Seventh Message Understanding Conference*. Morgan Kaufmann.

[Le, 2004] Le, Z. (2004). Maximum entropy modeling toolkit. `http://www.nlplab.cn/zhangle/`.

[Malouf, 2002a] Malouf, R. (2002a). A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of CoNLL-2002*, pages 49–55. Taipei, Taiwan.

[Malouf, 2002b] Malouf, R. (2002b). Markov models for language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pages 187–190. Taipei, Taiwan.

[Mayfield et al., 2003] Mayfield, J., McNamee, P., and Piatko, C. (2003). Named entity recognition using hundreds of thousands of features. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL-2003*, pages 184–187. Edmonton, Canada.

[McCarthy, 2003] McCarthy, J. (2003). *What is Artificial Intelligence?* `http://www-formal.stanford.edu/jmc/`, Computer Science Department, Stanford University. Last visited: February 19th, 2004.

[McNamee and Mayfield, 2002] McNamee, P. and Mayfield, J. (2002). Entity extraction without language-specific resources. In *Proceedings of CoNLL-2002*, pages 183–186. Taipei, Taiwan.

[Mikheev et al., 1999] Mikheev, A., Moens, M., and Grover, C. (1999). Named entity recognition without gazetteers. In *Proceedings of ACL'99*, pages 1–8. Bergen, Norway.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Book Co., Singapore, international edition.

[Munro et al., 2003] Munro, R., Ler, D., and Patrick, J. (2003). Meta-learning orthographic and contextual models for language independent named entity recognition. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL-2003*, pages 192–195. Edmonton, Canada.

[Paliouras et al., 2000] Paliouras, G., Karkaletsis, V., and Spyropoulos, C. (2000). Learning decision trees for named-entity recognition and classification. In *Proceedings of the Workshop "Machine Learning for Information Extraction"*. European Conference in Artificial Intelligence, Berlin, Germany.

[Palmer and Day, 1997] Palmer, D. D. and Day, D. S. (1997). A statistical profile of the named entity task. Technical report, The MITRE Corporation, Bedford, MA, USA.

[Pudil et al., 1994] Pudil, P., Novovičová, J., and Kittler, J. (1994). Floating search methods in feature selection. *Pattern Recognition Letters*, 15(11):1119–1125.

[Ratnaparkhi, 1997] Ratnaparkhi, A. (1997). A simple introduction to maximum entropy models for natural language processing. Technical Report 97-08, Institute for Research in Cognitive Science, University of Pennsylvania.

[Rennie, 2004] Rennie, J. D. M. (2004). Derivation of the F-measure. `http://people.csail.mit.edu/u/j/jrennie/public_html/writing/fmeasure.pdf`.

[Reunanen, 2003] Reunanen, J. (2003). Overfitting in making comparisons between variable selection methods. *Journal of Machine Learning Research*, 3:1371–1382.

[Sloman, 1998] Sloman, A. (1998). *What is Artificial Intelligence?* School of Computer Science, University of Birmingham. `http://www.cs.bham.ac.uk/~axs/misc/aiforschools.html` Last visited: February 18th, 2004.

[Somol et al., 1999] Somol, P., Novovičová, J., Pudil, P., and Paclík, P. (1999). Adaptive Floating Search Methods in Feature Selection. *Pattern Recognition Letters*, 20(11-13):1157–1163.

[StatSoft, 2003] StatSoft (2003). *Data Mining Techniques*. StatSoft Inc. `http://www.statsoftinc.com/textbook/stdatmin.html` Last visited: May 3rd, 2004.

[Tjong Kim Sang, 2002a] Tjong Kim Sang, E. F. (2002a). Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pages 155–158. Taipei, Taiwan.

[Tjong Kim Sang, 2002b] Tjong Kim Sang, E. F. (2002b). Memory-based named entity recognition. In *Proceedings of CoNLL-2002*, pages 203–206. Taipei, Taiwan.

[Tjong Kim Sang, 2003] Tjong Kim Sang, E. F. (2003). Improving machine learning results with filter rules. Presented at 2003 Atila meeting in Antwerp, Belgium. `http://lcg-www.uia.ac.be/~erikt/talks/`.

[Van den Bosch, 2004] Van den Bosch, A. (2004). Wrapped Progressive Sampling Search for Optimizing Learning Algorithm Parameters. In Schomaker, L., Verbrugge, R., and Taatgen, N., editors, *Proceedings of the Sixteenth Belgian-Dutch Artificial Intelligence Conference*, Groningen. To appear.

[Van den Bosch et al., 2004] Van den Bosch, A., ansupplementd Walter Daelemans, S. C., Hendrickx, I., and Tjong Kim Sang, E. (2004). Memory-based semantic role labeling: Optimizing features, algorithm, and output. In *Proceedings of CoNLL-2004*, pages 102–105. Boston, MA, USA.

[Van Rijsbergen, 1975] Van Rijsbergen, C. J. (1975). *Information Retrieval*. Butterworths, London, 2nd edition.

[Veenstra, 1998] Veenstra, J. (1998). Fast NP chunking using memory-based learning techniques. In *Proceedings of Benelearn 1998*, pages 71–79.

[Veenstra and Tjong Kim Sang, 1999] Veenstra, J. and Tjong Kim Sang, E. F. (1999). Representing Text Chunks. In *9th Conference of the European Chapter of the Association for Computational Linguistics*, pages 173–179, University of Bergen, Bergen, Norway.

[Winston, 1992] Winston, P. H. (1992). *Artificial Intelligence*. Addison-Wesley Publishing Company, 3rd edition.

[Wu et al., 2003] Wu, D., Ngai, G., and Carpuat, M. (2003). A stacked, voted, stacked model for named entity recognition. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL-2003*, pages 200–203. Edmonton, Canada.

[Wu et al., 2002] Wu, D., Ngai, G., Carpuat, M., Larsen, J., and Yang, Y. (2002). Boosting for named entity recognition. In *Proceedings of CoNLL-2002*, pages 195–198. Taipei, Taiwan.