A Named Entity Recognition System for Dutch

Fien De Meulder, Walter Daelemans and Véronique Hoste

April 25, 2002

Abstract

We describe a Named Entity Recognition system for Dutch that combines gazetteers, hand-crafted rules, and machine learning on the basis of seed material. We used gazetteers and a corpus to construct training material for Ripper, a rule learner. Instead of using Ripper to train a complete system, we used many different runs of Ripper in order to derive rules which we then interpreted and implemented in our own, hand-crafted system. This speeded up the building of a hand-crafted system, and allowed us to use many different rule sets in order to improve performance. We discuss the advantages of using machine learning software as a tool in knowledge acquisition, and evaluate the resulting system for Dutch.

1 Introduction

Named Entity Recognition (NER) is the problem of identifying different kinds of names in texts. For this system, we limited ourselves to finding the names of persons, companies, and locations. But this could later be extended to include names of organizations, and even different kinds of numerical expressions (e.g. dates and weights). The task we set ourselves can be illustrated by the following example:

> [..], Lt. Governor ¡person¿Mary Donohue¡¿ today announced that ¡company¿Bristol-Myers Squibb Company¡¿ has approved the expansion of its existing facilities in ¡location¿East Syracuse¡¿.

There are different ways in which the problem of NER has been solved in the field of computational linguistics. Many systems use a combination of **hand-crafted** sets of rules, in combination with gazetteers, or lists, of names, and grammars of names. There are many different hand-crafted systems for NER, such as Appelt and Martin (1999), Black, Rinaldi, and Mowatt (1998), Wacholder, Ravin, and Choi (1997), and Wakao, Gaizauskas, and Wilks (1996). The main disadvantage in hand-crafting such a system is that the development requires a lot of expertise and is very time-consuming: huge gazetteers need to be compiled, or many context rules need to be thought up. These rules then all have to be hand-coded. The system also needs a lot of testing and tweaking. According to Appelt and Martin (1999), the advantage of a hand-crafted system is that changes are quickly and easily implemented. One module can easily be changed, while a statistical approach might require the annotation of new training material, followed by another training phase. Even though Mikheev, Moens and Grover (1999a, 1999b) use a combination of hand-crafted rules and a statistical approach, we used them as our main inspiration for the hand-crafted part of our system only. Their sure-fire rules served as an inspiration for our rule component, and their partial match for our reprocessing steps.

Proceedings CLIN 2001, Rodopi, 77-88, 2002

The **statistical** approach to NER, on the other hand, looks at a combination of context features and internal features of words to be classified as NEs by some statistical model. Some of the main statistical approaches are Bikel et al. (1997), Borthwick et al. (1998a), Borthwick et al. (1998b), and Miller et al. (1999). All of these systems learn which features are important in NER. The advantage is that rules do not all have to be hand-coded any more, and that less tweaking of the system is necessary, because the computer will itself learn the order in which rules should be applied - i.e. the order of importance of context features. All systems cited above were Hidden Markov Model or maximum entropy learners. The disadvantage with this approach is that vast amounts of hand-annotated training data need to be produced. An additional disadvantage is that all statistical methods need some kind of post-processing step to generalize tags found for a type to all that type's tokens.

In our approach, we combined the advantages of the hand-crafted systems and the statistical systems.

Our system has two main components:

- In the first part, the system uses gazetteers or lists of names, combined with very simple grammars of names. Words are checked for membership of a class by looking them up in lists. The grammars are used to recognize strings of words as names.

- In the second part of the system, context rules are used to detect additional Named Entities. These rules were found by using the machine learning algorithm Ripper (Cohen 1995). These rules were then integrated into the rest of the system.

In the following Section we describe the hand-crafting component and rule-learning component of our NER system in more detail. Section 3 gives a global overview of our system architecture. In Section 4 we report on the results of our NE recognizer on a hold-out test set. In Section 5 we name some shortcomings of the current system and propose possible solutions. We end with some concluding remarks.

2 Combining hand-crafting and rule learning to build our system

Our NER system is hand-coded, but includes information learned by a machine learning technique which makes use of seed material. The hoped-for advantages were: on the one hand, a greater precision (because of the intervention of a human interpreter who can dismiss nonsense rules proposed by the learning algorithm), and, on the other hand, less time spent on designing the system (only the easy and well-described grammars of words were put into code by ourselves, while all the context rules were taken straight from the output of the machine learning algorithm).

First Names	4,931
Small Surname Words	12
Company Names	1,262
Words before Companies	62
Words after Companies	297
Place Names	3,284
Words before Place Names	9

Table 1: Size of the gazetteers which we used in our NE recognition system.

2.1 Gazetteers and grammars of names

The first component of the system recognizes NEs, using only lists of names (or **gazetteers**), and grammars of names. We used 7 gazetteers (see Table 1).

For **person** names, we used a list of first names, and one of words which commonly appear in surnames (eg. *de* for Dutch names, and *di* or Italian names). We did not use a list of surnames, because there are just too many of them to be able to make a good list. It is also not too hard to identify a lot of surnames to use as seed names, as we shall discuss later.

For **companies**, we compiled a list of the most important companies worldwide and in Belgium. We also manually compiled a list of words which often appear at the start of company names (eg. *International*), and a list of words which often appear at the end of company names (eg. *Limited*). These last two lists were necessary in order to keep the size of the gazetteer of companies down. If the system could not use this list, all partial orders of company names would also have to be stored in the list of company names. For example, just now we have the name *Microsoft* in our list of company names, and the words *Corporation and Corp.* in our list of words which can go after company names. These last two words can therefore be used for any corporation name ever encountered, while only one entry is needed for Microsoft in the gazetteer.

Finally, we compiled a list of international and Belgian **location** names, as well as a list of words which often appear before locations (eg. *South*, *New*). This had the added advantage that if a place like *Hamphshire* was included in the list of locations, *New Hampshire* would also automatically be recognized.

All lists were checked so that they did not contain any names which could be ambiguous between the different categories, or any names that could be interpreted as common nouns.

In addition to the gazetteers, we also integrated a simple **grammar of names**. Names have quite predictable structures. For example, if a first name appears with a capitalized word straight after it, chances are that this second word will be a surname. For Dutch names, the picture is slightly more complicated, because some words can appear in surnames without being capitalized (eg. *de*, *van*). However, none of these options are that hard to analyse, and regularities are easy to put into formalisms. For person names, we allowed a first name, one middle name, up to two small words, and then up to two capitalized words as a surname. For

companies: (optionally) a sequence of words which appear before companies + a word/phrase from the company list + (optionally) a sequence of words which go after companies = a company name. Considered as a location was: (optionally) one word which can precede a location + a word from the locations list.

We now brought the gazetteers and grammars together, and wrote a NER system which basically only recognized the companies and locations which were stored in the lists (plus extra words before and after). This system was enriched with a reprocessing step for the person names. The original program recognized person names if they started with a first name that could be found in the gazetteer of first names. Because these surnames might be repeated on their own elsewhere in the text, we introduced this reprocessing step into the system so that any surnames found after first names will also be recognized elsewhere in the text. This hugely increases the number of person names found in newspaper text.

2.2 Towards context rules

After the relatively straightforward step of building this name recognizer, we wanted to make it more 'intelligent' by learning context rules, and introducing these back into our system. We did this by creating seed material from Dutch newspaper text, and introducing this material as input into a rule learner, Ripper (Cohen 1995, Cohen 1996).

2.2.1 Use of seed material

Using seed material gets around the problems concerned with generating hand-annotated training data. For named entities, this approach was used already by Collins and Singer (1999), Cucerzan and Yarowsky (1999), and Buchholz and van den Bosch (2000). This method works by creating training material using the lists (e.g. by collecting the contexts where a word in the list occurs). This training material is then used to learn how to find those entities which could not be found in any of the lists.[1] Of the English NE finders, Cucerzan and Yarowsky (1999) used seed lists of between 40 and 300 words, whereas Collins and Singer (1999) surprisingly used only 7 seed rules. This shows that only a little bit of information is needed to get quite good accuracy (up to and accuracy of 91% in the case of Collins and Singer (1999)).[2] The Dutch approach (Buchholz and van den Bosch 2000) was to compare their system with small gazetteers (100 per class) to the performance of their system with extensive lists (53,065 names in total). They found that precision was improved when using smaller lists, and this with only a slight reduction in recall. Their overall accuracy was reported at 71% (on tokens).

[1] In some approaches, these new instances will then again be used as training material, starting an iterative process, which should lead to more and better results each time. We only implemented one such iteration.

[2] All accounts of NER systems seem to suggest that lists of very frequently occurring NEs work better than long lists of rare names. In the case of the use of seed material, it also seems that extra rounds of boosting do more than making lists of NEs longer ever could.

The potential advantages of this kind of seed approach are considerable: lists are very easy to make, and can be found on the internet as well.

Using lists of names, it is relatively easy to build an NER system with very high precision. This means that most of the named entities it finds will be correctly identified. However, this kind of system will typically have very low recall, so it will actually identify too few of the entities you would like to find. You can now use those few very good instances which you did identify in order to learn how to find the ones that got away. The way we did this was by using the very simple gazetteer look-up system to tag a large corpus. We then used the NEs identified to construct instances for the Ripper rule-learner. Ripper then learned context rules from these training instances. These training instances were then used to find the other NEs (those that were not included in our gazetteers). Introducing these context rules made for a big improvement in recall, while not affecting precision too dramatically.

2.2.2 Data

We used 250 days' worth of the Flemish financial newspaper *De Financieel Economische Tijd* (FET). This amounted to 13,951,459 tokens. This paper was chosen because of its economic orientation, meaning that it has more mentions of company names, thus providing us with more training data for this category than a more general paper would. Preprocessing was needed on the text. We first needed to strip it of its HTML tagging. Afterwards, it was tokenized and tagged with parts of speech using the memory-based tagger MBT (Daelemans et al. 1996), trained on the Dutch Eindhoven corpus. These tags were then available as features for the rule learner.

These data were processed with the simple NE tagger built so far. 270,899 NEs were extracted, of which 90,902 were tagged as person names, 80,157 as companies, and 99,840 as locations. Every portion of the text with a recognized NE in it was now turned into a training instance for the learning algorithm. Originally, we used an array of 75 features, but these were gradually reduced to leave only the 5 most important ones. The word before the NE turned out to be the most important feature. Using an idea from Mikheev, Moens, and Grover (1996b), we also added a feature to include nouns which appear in apposition to NEs, positioned after the NE. The following is an example of a regular expression used to find the noun (N), which was then used as a feature value:

```
,?      een?  Adj*  N
is?     het?
        de?
```

2.2.3 Learning context rules

We then used these data as training material for the Ripper (Cohen 1995) learning algorithm. Ripper is a member of the family of rule induction algorithms. Ripper's hypothesis is expressed as a set of if-then rules. Before learning, Ripper first

heuristically orders the classes. After arranging the classes, ripper finds rules to distinguish between the classes. The final class becomes the default class. We used Ripper as a knowledge acquisition device: we made it produce rules to structure our data, then read and interpreted them, and reused them in a separate module in our hand-coded system. All the rules which Ripper came up with (in the best runs) were read and interpreted by a human programmer, who took all the rules that made sense and put them in a new module for our NE recognizer. The following is an example of the rules which Ripper came up with. The words in bold were found to be indicators of NEs by the Ripper algorithm.

> De **resultaten van** Axa Royale Belge (The **results of** Axa Royal Belge)
> VEV-**voorzitter** Karel Vinck (VEV **chairman** Karel Vinck)
> Barton Biggs, de grote **baas** van (Barton Biggs, the big **boss** of)

By varying the Ripper algorithm parameter settings, we managed to get better rules for different classes. We did not have to choose one of the runs over the others, but could simply reap the benefits from all, by putting all good rules together in our new module. Another fruitful strategy was to manipulate the training data. We varied the proportions of each of the NE classes, so that we got the best rules for each class. These rules were then again all combined to make a far stronger rule component than any individual Ripper rule set on its own could. Ripper also found some 'bad' rules. This was partly due to mistakes made by the gazetteer look-up system, and partly due to the lack of negative examples in the training data. These bad rules, however, were easily dismissed in the manual post-processing step.

3 System architecture

Schematically, the final system looked like Figure 1.

- The first step is that of **gazetteer look-up**. This is the same module as the one that was used to produce the seed rules earlier.
- The second step is the **reprocessing of surnames**. This was also already done when making seed rules: the surnames found after first names from the gazetteers are also tagged when they appear on their own elsewhere in a document.
- The third step, that of the **context rules**, is new. This is the collection of rules which Ripper found. These are context rules, mainly focused on nouns found in apposition to NEs. Another kind of rule involves nouns separated from an NE by a preposition (eg. the capitalized words after *aandelen van* (*shares in*) most likely refer to a company). These rules are implemented by simple regular expressions.
- Finally, in the fourth step, another **reprocessing round** follows. Person names and company names which were identified in the previous step are

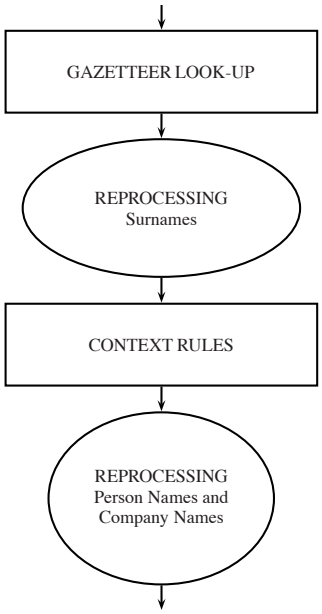

Figure 1: Sketch of the system.

now tagged elsewhere in the text. Not only full names are tagged elsewhere, but so are different partial orders of NE phrases. (Eg. if *Joe Bloggs* is recognized as a person, so will *Bloggs* on its own. The same goes for *Acme Inc*. and *Acme*.) This final step is important, because it means that it is sufficient for the context rules to only recognize one single token for each type of NE. The system will then itself generalize over all other tokens of the same type.

This full cycle of four steps is applied to each newspaper article separately. The small gazetteers of frequently occurring names are used as before. It might seem to make sense to update these with the new NEs found in the course of the runs of the system, but this is not done, in order to reduce errors. Each article is also processed separately: when NEs are found, they are tagged as the same kind of NE elsewhere in the article, but not elsewhere in the whole of the document. There is a logic to this: when a new NE is introduced, especially a rarer one (which is also less likely to occur in the gazetteers), this will often be reflected in the language surrounding this new name. NEs are often 'introduced' to people. For example, a lesser known minister will first be introduced as *Minister of Culture and Sports Jane Smith*, but she will later just be referred to as *Smith*. If the context rules are good, our system should catch the first occurrence of this NE, and reprocessing will then correctly

tag all other references to the same person in the same article.It is sensible to generalize the tagging of types to tokens within articles, but it is not sensible to generalize over a whole newspaper, or a full corpus to be tagged. For example, if *Permeke* (the company) is identified in an article, it is temporarily remembered and identified as a company elsewhere in the same article. If it was then also stored for use in other articles, the artist *Permeke* may be wrongly identified as a company later. The same goes for NEs which have the same form as a common noun, eg. *Apple*. If this word is kept in a gazetteer permanently, it may be that a mention of a piece of fruit is wrongly tagged as referring to a company. within the same article, this is far less likely to occur. [3]

The final system was implemented in Perl. Its structure stayed modular, making it very easy to change in the future. For example, the name grammars could very easily be changed to allow for names from different countries than the ones already included. The gazetteers could easily be updated, if new names replace older popular names. If the system was applied to a different domain (a different kind of text), then the grammars of names and gazetteers could be kept the same, as well as all reprocessing steps. The only thing needed in this case would be a quick round of making new training material and training Ripper on it. In the modular system, only the context rules would then need to be adapted.

4 Experimental results

In order to test the performance of our NER system, we extracted a hold-out test file extracted from the FET newspaper. It had 32,826 tokens and was manually tagged for NEs. The test file contained 469 person NEs, 342 companies, and 279 locations. Scores were computed per NE class (Person, company and Location), and the measures used were Precision (of the tags allocated by the system, how many were right), Recall (of the tags the system should have found, how many did it spot), and F-score (a combination of Recall and Precision). The scores on the test set are displayed in Table 2. The results are broken down into the different categories. We measured the performance of the system with only the first module working, then the first two modules together, and so on.

Table 2 shows that:

- the **person** category started off performing well in the first module. Precision was high, and recall not too bad (at 41.59%). The F-score for this category consistently rises with each module, but there is a marked drop in precision (of more than 15%).
- The **company** tag is consistently assigned with high precision, but recall is quite disastrously low (ranging between 23.17% and 37.70%). There is a marked improvement in recall, however, between modules 1 and 4, so the way to improve the performance in this area may well be through a

[3] There may be an argument for adding non-ambiguous very frequently occurring names, but no such mechanism has been implemented in the system so far.

		Precision	Recall	F-score
1 module	Person	90.19%	41.59%	56.93%
	Company	98.47%	23.17%	37.51%
	Location	76.19%	66.90%	71.24%
2 modules	Person	77.88%	52.37%	62.63%
	Company	98.47%	23.17%	37.51%
	Location	75.61%	64.81%	69.79%
3 modules	Person	76.13%	65.30%	70.30%
	Company	94.44%	26.53%	41.42%
	Location	74.74%	74.22%	74.48%
4 modules	Person	74.63%	75.43%	75.03%
	Company	91.81%	37.70%	53.45%
	Location	76.34%	74.22%	75.27%

Table 2: Scores of our NE recognition system on a hold-out test set.

combination of extending the gazetteers and more training runs for Ripper in order to find more context rules.

- For the category of **location** NEs, it is remarkable to see how little influence context rules have. Even though precision is not very good throughout, location starts off in module 1 with the highest F-score of all the categories (71.24%), and it does not improve much beyond that with the addition of context rules (a mere 4% improvement). This indicates that Ripper was not able to formulate good context rules for the location NEs, which is also shown by the fact that reprocessing locations in the last module actually worsened performance in the design stages of the system (so this reprocessing step was not kept in the final design). The main way to improve locations therefore seems to be through the extension of the gazetteer. Precision will probably also need to be tackled by re-examining the gazetteer, and checking it again for possible ambiguous words.

A methodologically correct quantitative comparison with other NE recognition systems is impossible because of differences in target language, text style and annotation conventions. The results reported below are merely an indication of the performance of other NE recognizers. For English, results for the hand-crafted systems are reported to hover around an F-score (combined Recall and Precision) of 90%. The results of the statistical systems ranged from an F-score in the 80s to and F-score in the middle 90s (Borthwick et al. (1998b) using a combination of different NE systems). The combined approach of Mikheev, Grover and Moens (1999a, 1999b) had an F-score of 90% and higher. For Dutch, Buchholz and van den Bosch (2000) reported an overall accuracy on types of 71%.

5 Problems and future research

The currently implemented system still suffers from some shortcomings. One problem with our system is that it is not very flexible. For example, the grammar of names is very rigidly applied, so it deals badly with events like spelling mistakes, such as *[..] minister van Ambtenarenzaken Luc* dan *den Bossche* (Eng.: *[..] Minister of Civil Service Luc* dan *den Bossche*). The grammar of names will not recognize this as a name, because it says *dan* rather than *van*.) A more statistical approach might be trained to deal with this kind of event (if it occurs in regularly), but our system will ruthlessly reject this name.

Another problem is the very narrow scope of the context rules. This component should in the future be taught to look much further afield for interesting nouns which might signal the presence of an NE. Our system, for example, does not yet recognize the noun in the following sentence: *De* voorzitter *van het Duitse Federale Bureau voor de Statistiek, Johann Hahlen* (Eng.: *The* chairman *of the German Federal Bureau of Statistics, Johann Hahlen*). Better use of parts of speech tags may well help to solve this kind of problem.

We intend to introduce two further modules into this system. A first module would allow the system to deal with conjunctions. Consider the following example: *China, Roemenië en Spanje [..]* (Eng.: *China, Romania and Spain [..]*). If two of these words are recognized as locations, it is extremely likely that the third one is, too. However, our system does not yet know how to handle this kind of construction. The second major shortcoming of our system is that it does not yet use its gazetteers of company and place name indicators sufficiently. *Nicos Life* will not be recognized as a company if *Nicos* is not included in the list of companies, and no suggestive context is found around it, even though *Life* is included in the gazetteer of words which go after company names. The system should be taught to recognize this as a potential company in the future.

Finally, we would like to test our intuition that our use of Ripper rules makes this system easily adaptable for new domains. If we use the simple gazetteer look-up system to make training material in a very different domain, we expect the output of Ripper rules to look different as well. A more general newspaper, for example, will probably give less context rules to find person names with business functions in them, and probably more context rules with sports terms, or entertainment functions.

6 Conclusion

We showed that also in a handcrafting approach to language engineering, Machine Learning can play a role as a tool in knowledge acquisition and knowledge engineering. Our pragmatic use of the rule induction system Ripper combined with a seed approach to collecting training data, enabled us to build a reasonably good system with no time wasted on manual annotation and painstaking design of context rules. The only thing which still took up some time in this method, is the different training runs Ripper needed to go through in order to find the best rules.

Setting up an approach to controlling this process in more detail would be a useful topic for future research.

Using this methodology, we succeeded in quickly building a system for NER in Dutch with at least for Dutch state-of-the-art precision and recall.

References

Appelt, D. E. and Martin, D. L.(1999), Named entity recognition in speech: Approach and results using the textpro system, *Proceedings of the DARPA Broadcast News Workshop*, pp. 51–56.

Bikel, D. M., Miller, S., Schwartz, R. and Weischedel, R.(1997), Nymble: a high-performance learning name-finder, *Proceedings of the 5th Conference on Applied Natural Language Processing*, pp. 194–201.

Black, W. J., Rinaldi, F. and Mowatt, D.(1998), Facile: Description of the ne system used for muc-7, *Proceedings of the 7th Message Understanding Conference*.

Borthwick, A., Sterling, J., Agichtein, E. and Grishman, R.(1998a), Exploiting diverse knowledge sources via maximum entropy in named entity recognition, *Proceedings of the 6th Workshop on Very Large Corpora*.

Borthwick, A., Sterling, J., Agichtein, E. and Grishman, R.(1998b), Nyu: Description of the mene named entity system as used in muc-7, *Proceedings of the 7th Message Understanding Conference*.

Buchholz, S. and van den Bosch, A.(2000), Integrating seed names and n-grams for a named entity list and classifier, *Proceedings of LREC*, pp. 1215–1221.

Cohen, W. W.(1995), Fast effective rule induction, *Machine Learning: Proceedings of the 12th International Conference*, pp. 115–123.

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

Cucerzan, S. and Yarowsky, D.(1999), Language independent named entity recognition combining morphological and contextual evidence, *Proceedings of the Workshop on Very Large Corpora at the Conference on Empirical Methods in NLP*.

Daelemans, W., Zavrel, J., Berck, P. and Gillis, S.(1996), Mbt: A memory-based part of speech tagger-generator, *in* E. Ejerhed and I. Dagan (eds), *Fourth Workshop on Very Large Corpora*, pp. 14–27.

Mikheev, A., Grover, C. and Moens, M.(1999a), Xml tools and architecture for named entity recognition, *Markup Languages* **1**(3), 89–113.

Mikheev, A., Moens, M. and Grover, C.(1999b), Named entity recognition without gazetteers, *Proceedings of the 10th Conference of the European Chapter of the ACL*.

Miller, D., Schwartz, R., Weishedel, R. and Stone, R.(1999), Named entity extraction from broadcast news, *Proceedings of DARPA Broadcast News Workshop*.

Wacholder, N., Ravin, Y. and Choi, M.(1997), Disambiguation of proper names in text, *Proceedings of the 5th Applied NLP Conference*, pp. 202–208.

Wakao, T., Gaizauskas, R. and Wilks, Y.(1996), Evaluation of an algorithm for the recognition and classification of proper names, *Proceedings of the 16th International Conference on Computational Linguistics*, pp. 418–423.