# Integrating seed names and ngrams for a named entity list and classifier

**Sabine Buchholz and Antal van den Bosch**

ILK / Computational Linguistics
Tilburg University, P.O. Box 90153, NL-5000 LE Tilburg, The Netherlands
{S.Buchholz,Antal.vdnBosch}@kub.nl, http://ilk.kub.nl

**Abstract**

We present a method for building a named-entity list and machine-learned named-entity classifier from a corpus of Dutch newspaper text, a rule-based named entity recognizer, and labeled seed name lists taken from the internet. The seed names, labeled either as PERSON, LOCATION, ORGANIZATION, or ADJECTIVAL name, are looked up in a 83-million word corpus, and their immediate contexts are stored as instances of their label. The latter 8-grams are used by a memory-based machine learning algorithm that, after training, (i) can produce high-precision labeling of instances to be added to the seed lists, and (ii) more generally labels new, unseen names. Unlabeled named-entity types are labeled with a precision of 61 % and a recall of 56 %. On free text, named-entity token labeling accuracy is 71 %.

## 1. Introduction

The task of named entity (NE) classification consists of mapping proper names to their semantic class. Commonly used classes are PERSON, LOCATION and ORGANIZATION. NE classification is crucial for information extraction, but might also be useful as a preprocessing step to parsing, or in search engines (if users can specify which class of an ambiguous type they are interested in).

Conceptually, mapping can take place at two different levels: tokens and types. A token is an occurrence of a proper name in text. As such, it is always accompanied by context, and it should always have a unique class. A type, on the other hand, is an abstract entity,i.e. the name itself, independent of any context. Most types still have a unique or at least dominant class. Even if it is true that e.g. a film might be called *Amsterdam*, it is still useful to know that *Amsterdam* normally refers to a LOCATION.

Types and tokens of the same name are of course not independent. To know the class ofa type, we have to look at many of its tokens. to classify tokens, many NE classification systems make use of gazetteers which contain large lists of classified types.

In this paper, we present a machine learning method to classify tokens and types of Dutch named entities. We evaluate both parts of the method through comparison with hand-annotated test sets. Train and test sets are extracted from a tokenized Dutch newspaper corpus. Classification accuracy of tokens is 71%, and of types 66.6%. Precision and recall of types is 60.9% and 56% respectively.

A prerequisite to NE classification is NE segmentation/recognition, i.e. finding those strings in a text that constitute a proper name. We implemented a rather straightforward NE recognizer and evaluated this, too.

We start with an overview of related research in Section 2.. Then we describe the prerequisites of our method, viz. the Dutch corpus, the basic named-entity extractor and the seed name lists, in Section 3.. Section 4. describes the experiments we performed in producing high-precision additions to our seed lists, as well as the experiments on named-entity classification. We state our conclusion and issues for further research in Section 6..

## 2. Related Research

(Cucerzan and Yarowsky, 1999) present a language-independent approach and test it for Romanian, English, Greek, Turkish and Hindi. Their system starts with small hand-built seed lists (150 to 300 seed words, depending on the language) and classifies tokens through EM-style bootstrapping, using word internal and contextual clues. Three classes are distinguished: first name, last name, and place. The baseline for this task for Romanian is 98.67% precision and 34.01% recall, yielding an F-measure of 50.58. Final system performance is 76.95% precision and 64.99% recall (F-measure 70.47). (Collins and Singer, 1999)

## 3. Method

### 3.1. Overview

As the class of a type can be inferred from the class(es) of its tokens, the basic module in our approach is a token classifier. It is realized by IGTree (Daelemans et al., 1997), a memory-based machine learning algorithm that can handle symbolic data (cf. Section 3.6.).

Training material is created mostly automatically by using seed lists gathered from the internet. Each token in the corpus of the types on the seed lists normally provides one training example (train instance). If a type has more than one class (i.e. it is ambiguous) several instances are produced.

The features used are the context words to the left and the right of the name token. Thus in contrast to (Cucerzan and Yarowsky, 1999), we do not use name internal information (e.g. substrings like *Mr.* or *Inc.* yet. Also only local information is used. Thus if e.g. a text reads "*Mr. John Smith won the contest. Smith said ...*", the two tokens and types (*Mr. John Smith* and *Smith* are classified independently. These restrictions make the method simple and fast.

Once all tokens of a type have been classified, the class of the type is the majority class of its tokens. In this step, thresholds can be used to make the classifier abstain from classification if the majority is not "convincing" enough.

In the next sections, we briefly introduce the corpus, the tokenizer, the seed lists, the method of extracting instances,

and the IGTree classifier. We will then describe how token and type classification proceeds, how the test sets were made, and how we evaluated the two parts of the method.

### 3.2. The corpus

The Dutch corpus used in our experiment is a subset of the ILK corpus, a collection of Dutch newspaper text archives. The subset consists of 160,626 articles (4,432,999 sentences, 83,383,018 words) from three regional newspapers (*Gelderlander*, 1996; *Brabants Dagblad* and *Eindhovens Dagblad*, both 1998) and one central news agency archive spanning 1985–1991. All texts were collated and article boundaries removed.

### 3.3. The tokenizer

Tokenization serves two purposes. First, it separates punctuation from words, to be able to recognize potential named entities in sentences like *Before we met Tom Jones, we went out to dinner*, where the comma should be separated from *Jones*. Second, tokenization also decides whether a period is an abbreviation period or sentence final punctuation; a non-trivial problem (**?**) that we tackled using a hand-edited semi-automatically generated abbreviation list for Dutch. The tokenizer does a good job in the above two goals, and can use its output also to segment sentences [1]

### 3.4. The seed lists

The seed lists for the commonly used classes PERSON, LOCATION and ORGANIZATION were found by manually searching the internet. Table 1 gives an overview. Of the total of 53,065 names, 214 have two classes, and 3 (*Dale, Horn, Zurich*) have all three classes.

The guidelines we adopted for annotating the test sets were the following: PERSON includes entities like *Shiva*, but not bands like *The Who*. LOCATION includes continents, regions, countries, districts, cities, villages etc. but not streets, building and rivers. ORGANIZATION includes companies, political parties, non-profit organizations etc. Note that the internet lists do not contain examples for all subclasses of a class. In a given application, it might be useful to chose the precise limits of classes differently.

During annotation of the test set, we noted another frequently occurring class of capitalized words: adjectives derived from names, like *English*[2]. Although these words do not directly denote named entities in a strict sense, they are nevertheless important for information extraction: a text that talks about "American politics" talks about "America" in a way. The seed list for the adjectival class was taken from the electronic Dutch dictionary CELEX (Baayen et al., 1993) from which we extracted all adjectives starting with a capital letter.

[1] Difficulties in sentence segmentation arise because of informally-marked (sub)section headings without periods, and article-introductory constructions like *Washington (Reuter) In Washington* . . ..

[2] In Dutch, separate forms exist for *the English=de Engelsen* and *English* as in *an English village* or *a book in English=Engels*

| | |
|---|---|
| PERSON | 717,512 |
| LOCATION | 494,683 |
| ADJECTIVAL | 210,581 |
| ORGANIZATION | 91,163 |

Table 2: Class distribution for train instances

Figure 1: Some train instances

### 3.5. Training instances

For each type that appears on any of the seed lists, we extracted all tokens together with a context of four words to the left and four words to the right from the corpus. [3] This yields 1,513,939 instances. The distribution of classes over instancs is shown in Table 2.

Figure 1 shows some example instances.

### 3.6. IGTree

The weights of the nine features are shown in Table 3.

As could be expected, the name itself (the word in focus) is the most important feature. This would mean that during classification of new tokens, the algorithm would first try to match on this feature. If the feature does not match, the default class would be assigned without any further feature matching. This is not what we want, as it amounts to list look-up only, and does not really classify the most interesting cases (those that were not on the lists). We therefore chose to let the algorithm ignore this feature during classification.

As can be seen from Table 3, the nearer the context word is to the name token, the more important it is for classification. At equal distance, words in front of the name are more important than those after it. The effect of this is that IGTree classifies a name token by using n-gram patterns centered around the name. By inspecting the IGTree built during classification, we can find interesting n-gram patterns. Table 4 shows a selection of patterns and their majority class.

The most general patterns (supported by most training instances) indicate that persons and places are normally not

[3] If a type has more than one class, instances are multiplied with the different classes.

| Feature | GainRatio |
|---|---|
| left-4 | 0.029 |
| left-3 | 0.038 |
| left-2 | 0.058 |
| left-1 | 0.132 |
| focus | 0.246 |
| right-1 | 0.095 |
| right-2 | 0.049 |
| right-3 | 0.032 |
| right-4 | 0.027 |

Table 3: Feature weights

| Class | Content | Number of names |
|---|---|---|
| PERSON | international male first names | 1,219 |
| PERSON | international female first names | 4,275 |
| PERSON | Dutch first names | 5,177 |
| PERSON | international last names | 31,821 |
| PERSON | Dutch last names | 806 |
| PERSON | total | 40,618 |
| LOCATION | Dutch cities/villages | 4,819 |
| LOCATION | English country names | 163 |
| LOCATION | Dutch country names | 231 |
| LOCATION | total | 5,135 |
| ORGANIZATION | Dutch non-profit organizations | 570 |
| ORGANIZATION | USA companies | 1,292 |
| ORGANIZATION | Dutch companies | 4,845 |
| ORGANIZATION | Dutch media | 783 |
| ORGANIZATION | total | 7,312 |
| ALL | total | 53,065 |

Table 1: Seed lists contents and sizes

| | | |
|---|---|---|
| een [FOCUS] vrachtwagen | a [FOCUS] truck | A |
| burgemeester van [FOCUS] , | mayor of [FOCUS] , | G |
| PvdA en [FOCUS] . | Socialist party and [FOCUS] . | O |
| staatssecretaris [FOCUS] | state secretary [FOCUS] | P |

Table 4: Some n-gram patterns represented in IGTree's decision tree, with English translation.

| | |
|---|---|
| PERSON | 348 |
| LOCATION | 260 |
| ADJECTIVAL | 182 |
| ORGANIZATION | 268 |

Table 5:

preceded by an article, whereas this is not unusual for organizations, e.g. *the Association for Computational Linguistics*. At the moment, it is impossible to use n-grams (n¿1) that span only one side of the context. However, this would clearly be useful.

## 4. Results

In this section we present the results on both named entity token labeling and type labeling. In addition, we will report on our experiences with the simple NE recognizer and the use of seed lists from internet versus hand-built lists.

### 4.1. Classifying tokens

To make a test set for the token classifier, we took the first 49 articles of the ANDA corpus, a central news agency archive spanning 1985–1991 (54,835 articles, 32,538,702 sentences 1,460,940 words), and manually segmented and classified all named entities of our four classes. This yields 1058 tokens. The class distribution is shown in Table 5.

To classify a token, we have to distinguish two cases: First, if the name is on one of the seed lists, we just adopt the corresponding class (list look-up).[4] With this method, 403 of the 1058 instances can be classified, with a precision of 90.8% and a recall of 34.6%.[5] Following (Cucerzan and Yarowsky, 1999), we take the baseline (forced classification) accuracy for this task to be the accuracy achieved by classifying everything on the seed lists by list look-up, and everything else as PERSON (the default, most frequent class). Baseline (forced classification) accuracy is 62%. Second, if a name is not on any seed list, it is classified by IGTree (cf. Subsection 3.6.), trained on the train instances described in Subsection 3.5.. As all instances are assigned one of the for classes by IGTree, it no longer makes sense to report precision and recall (as they would be identical), instead we use accuracy. Accuracy of the 655 instances classified by IGTree is 58.8%. %. Overall (forced classification) accuracy (taking together instances classified by list look-up and by IGTree) is 71% which is well above the baseline of 62%.

### 4.2. Classifying types

When evaluating token classification, it makes sense to also classify tokens whose type is on one of the seed lists. When classifying types, on the other hand, we are mainly

[4] If a name is on several lists, just one class is assigned. This happens with *Israel, India and Canada*, which, according to the lists, are person as well as location. Maybe the algorithm should better abstain.

[5] Precision means the percentage of classified instances that are classified correctly, recall means the percentage of all instances that are classified correctly.

| | |
|---|---|
| PERSON | 319 |
| LOCATION | 65 |
| ADJECTIVAL | 29 |
| ORGANIZATION | 193 |

Table 6:

interested in new types, in order to be able to add them to the lists once they are classified. To create a type test set, we let a simple NE recognizer (see Subsection 4.3.) extract all NE's from the corpus, producing a list of 662,034 names. Then all names that are on the seed lists are deleted. From this remaining subset, we randomly chose 910 of them turned out to be types of our four classes. Table 6 shows the class distribution. The baseline accuracy for this set is gained by classifying everything as the default class PERSON, which yields 52.3%.

To let our system classify these types, we have to make a detour through token classification. As for the train set, we extract all tokens from the corpus that contain one of the names on the type list. This yields 21,754 instanes. After classifying all instances with IGTree, we set the class of the type to be the majority class of all its tokens. This approach yields a (forced classification) accuracy of 66.6% (cf. the 52.3% baseline).

In the above evaluations, we over-simplified the task of NE classification. Even if we had a perfect NE recognizer, it would still recognize named entities that do not fall in any of our four classes. Indeed, our simple NE recognizer (see Section 4.3.) extracted 186 of those types, too. Examples are names of bands, streets, rivers, buildings, books, films, newspapers, reports, laws etc. We would want our NE classifier to abstain from classifying these instances, i.e. indirectly marking them as a OTHERNAME class.

To achieve this goal, we introduced two thresholds into our type classifier. As we explained in Section 3.6., each node in the IGTree carries a (default/majority) class and the class distribution from which the majority class was computed. Now, if the majority class has a majority of less than x% (according to the distribution), we change the node class to OTHERNAME. Let us call x the token threshold. Then, during the step from tokens to types, we again have a majority class and a distribution (of the classes of the tokens). Again, we change the (type) class to OTHERNAME if majority is below y% (y: type threshold).

To evaluate the types now classified as either PERSON, LOCATION, ADJECTIVE, ORGANIZATION or OTHERNAME, we again use precision and recall[6] Thus an OTHERNAME type classified as PERSON counts as a precision error, a PERSON type classified as a NONE counts as a recall error, and e.g. a LOCATION type classified as a PERSON type counts as a precision and recall error. Overall precision and recall crucially depends on the settings of the token and type thresholds. In general, the higher the threshold, the higher precision, but the lower recall. See Figure 2 for an overview of performance with different threshold settings.

Figure 2: Precision and recall depending on thresholds

In addition to overall precision and recall, we can now also compute precision and recall per class. We notice large differences between the classes in this respect.

### 4.3. Named entity recognition

The most simple NE recognizer just extracts each sequence of one or more capitalized words as a name (this works for English and Dutch but not all language even have capitalization). If the text is tokenized, the recognizer can treat capitalized words at the beginning of sentences differently. If we compute word form frequencies for the whole corpus, we can avoid extracting capitalized words at the beginning of sentences that occur with a lower case letter more often than with a upper case. Along the same reasoning we can determine that in the sentence beginning "*Maar Jan ...*" ( *But John ...* only *Jan* and neither *Maar* nor *Maar Jan* is a name. If we use a list of closed class words (e.g. from CELEX, (Baayen et al., 1993)), we can even filter out similar cases in which the beginning of the sentence is not clearly marked[7] A special problem is formed by last name prefixes (e.g. *van den*, cf. the second author's name) that are very common in Dutch and have to be written with a lower case letter if a first name precedes the last name, and with an upper case letter otherwise. We gathered a list of 40 such prefixes (including e.g. German *von* and Italian *di*). However not every sequence of capitalized words with a possible infix in between is a name. The common prefix *van* (as in *Vincent van Gogh*) also appears in constructions like *burgemeester Patijn van Amsterdam* (*mayor Patijn of Amsterdam*) where the whole is not a name. Using the internet lists, we therefore restricted the extraction of these constructions to cases in which the first part is known to be a first name, and the latter to be a last name. This leaves names like *Rio de Janeiro* unextracted.

We let a NE recognition program implementing these heuristics extract names from the corpus. In the process of building a type test set, we manually annotated 910 of the extracted strings. Of these, 118 are not names, but either normal (open class) nouns that are capitalized because they start a sentence, while the sentence start is not recognized because the previous sentence does not end in punctuation, or else parts of names like *Seven Years (in Tibet)* or *Association (for Computational Linguistics)*. 40 extracted strings contained in fact two names, either in constructions like *het Noord-Brabantse Alphen* where *Noord-Brabantse* is an ADJECTIVE, but *Alphen* is a LOCATION, or like *London/Paris* where the tokenizer failed to separate the two names, or else like *omdat Lubbers Kok ...* where one name directly follows another as adjacent subject and object. For the evaluation, we treated these double constructions as instances of the class of the second name.

[6]Precision means the percentage of instances classified as PERSON, LOCATION, ADJECTIVE, or ORGANIZATION that are classified correctly, recall means the percentage of all true PERSON, LOCATION, ADJECTIVE, or ORGANIZATION instances that are classified correctly.

[7]In our corpus, sentence after (sub)titles frequently show this problem.

### 4.4. Internet seed lists vs. hand-built lists

(Cucerzan and Yarowsky, 1999) show that small hand-crafted seed lists (e.g. 100 names per class) can already be useful for NE classification. They claim that making these small lists is not much of an effort. Typically the quality of these lists is very high. Alternatively, finding some seed lists on the internet is not much work either. This approach typically yields large, but somewhat noisy lists. We are therefore also interested in finding out what the effects of these different lists are.

We manually assembled lists of 100 names per class and repeated all of the experiments described above with these lists. As there are a lot less names on the seed lists now, the number of training instances is also less: 961,407 (vs.1,513,939 when using the internet lists). When classifying the token test set, less instances can be classified directly by list look-up: 310 (vs. 403). However, precision is much higher: 99.7%; recall is 29.2%. Baseline accuracy (list look-up, everything else default class PERSON) is comparable: 60.2% vs. 62% with the internet seed lists. Overall system accuracy however is lower: 64.4% vs. 71%. The same holds for type classification: accuracy is 45.6% vs. 66.6%, precision and recall are 47.8%/34.4% vs. 60.9%/56%. Thus in our case, large noisy internet lists proved more succesful than small high-quality hand-built lists.

A possible explanation comes from the distribution of classes in both train sets. In the train set generated with the internet lists, PERSON is the most frequent class, then LOCATION, ADJECTIVE and last ORGANIZATION. This is the same order as in the token test set. In the type test set, the order is PERSON, ORGANIZATION, LOCATION , ADJECTIVE, but PERSON still is the most frequent type. In the train set generated from the hand-built lists however, the order is LOCATION, ADJECTIVE, PERSON, ORGANIZATION. This means that whenever there is little or conflicting evidence about how to classify an instance, the algorithm will take LOCATION as default which is a worse choice than PERSON in most cases. Indeed, LOCATION has the highest recall butthe lowest precision when looking at individual classes.

## 5. Future Research

Evaluate using a threshold on tokens and NONE class token instances. Adding new types to lists.

## 6. Conclusion

## 7. References


Baayen, R. H., R. Piepenbrock, and H. van Rijn, 1993. *The CELEX lexical data base on CD-ROM*. Philadelphia, PA: Linguistic Data Consortium.

Collins, M. and Y. Singer, 1999. Unsupervised models for named entity classification. In P. Fung and J. Zhou (eds.), *Proceedings of EMNLP/VLC'99*. College Park, MD: ACL.

Cucerzan, S. and D. Yarowsky, 1999. Language independent named entity recognition combining morphological and contextual evidence. In P. Fung and J. Zhou (eds.), *Proceedings of EMNLP/VLC'99*. College Park, MD: ACL.

Daelemans, W., A. Van den Bosch, and A. Weijters, 1997. IGTree: using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*, 11:407–423.