Training a Named Entity Recognizer on the Web

David Urbansky, James A. Thom, Daniel Schuster, Alexander Schill

Dresden University of Technology
RMIT University
{david.urbansky,daniel.schuster,alexander.schill}@tu-dresden.de
james.thom@rmit.edu.au

Abstract. In this paper, we introduce an approach for training a Named Entity Recognizer (NER) from a set of seed entities on the web. Creating training data for NERs is tedious, time consuming, and becomes more difficult with a growing set of entity types that should be learned and recognized. Named Entity Recognition is a building block in natural language processing and is widely used in fields such as question answering, tagging, and information retrieval. Our NER can be trained on a set of entity names of different types and can be extended whenever a new entity type should be recognized. This feature increases the practical applications of the NER.

1 Introduction

Named Entity Recognition (NER) is an information extraction task that the Message Understanding Conference 6 (MUC-6) originally defined in 1996. The goal of the task was to identify six types of entities. These were names of people, organizations, geographic locations, times, currencies, and percentage expression in texts [12]. NER is an important task in the chain of natural language processing, and other techniques such as co-reference resolution, machine translation, language understanding, and question-answer finding build upon it. With the trend toward a more entity-centric search paradigm, NER will become even more important in the future. Essentially, there are three types of NER systems: Those that use hand-crafted rules, those that use supervised machine learning algorithms, and those that use unsupervised machine learning [23]. Hand-crafted rules are often more precise, but yield lower recall. Moreover, they require a lot of manual work. When there are annotated corpora available, the preferred technique is to use supervised machine learning. Often we do not have corpora to train a named entity recognizer, and compiling one is a very time consuming task. Consider an application that wants to detect mentions of mobile phone names in news feeds. The usual tagged corpora contain only four entity types – person, organization, location, and miscellaneous. Training the NER on such a corpus would not be beneficial since mobile phone names fall under the category miscellaneous, as would digital cameras, cars, movie names, etc. We would therefore need to label the mobile phone names in a corpus of news feed texts. If our application advances and we also want to recognize mentions of digital

cameras, we need to update that corpus and label all mentions of digital cameras as well. This scenario parallels the ambitions of the WebKnox research project [28] in which the goal is to extract unknown entities from the web.

In this paper we try to answer the following research questions:

- How much labeled data is necessary to train a well-performing NER?
- How well do state-of-the-art NERs perform on web texts recognizing many different entity types?
- How well does a NER perform when it is automatically trained on the web?

Our contributions are answers to these questions using two datasets, one with 4 entity types (CoNLL) and one with 22 entity types (TUDCS4). It is important to note that almost all related work limits its findings to a very small number of entity types as found in the MUC or CoNLL datasets. First, we analyze the performance of three state-of-the-art NERs based on the size of the training data and the number of entity types that need to be recognized. Second, we propose a technique to train a NER on a set of seeds automatically, and show how well this approach performs compared to supervised learning. Another contribution is the release of our dataset on the research platform Areca[1] to allow other researchers use the corpus. Furthermore, our developed Named Entity Recognizer and all related tool are available free for research purposes[2].

2 Related Work

There are three major approaches to named entity recognition. First, there are lexicon-based recognizers that can only detect entities that they have stored in their lexicon or gazetteer [21, 14]. Often there are ambiguous entities such as "Paris", which is both a city (and there are many cities called Paris) and a forename. Using lexicons is therefore not simple and matches need to be disambiguated. This approach is not of interest to us, however, since our goal is to recognize unknown entities. Second, there are hand-crafted rules for NERs [18, 6]. One such rule could be "the uppercase sequence of words after the token 'Mr.' is a name". Ripper [7] is a system used to create such if-then rules. Rule-based systems were mainly popular in the early days of NER research and resulted in high precision. This approach does not scale well, however, since rules need to be created by experts and must be maintained for every new entity type that should be recognized [19]. For instance, the example rule for detecting person names using the prefix "Mr." has to be refined when the NER is supposed to differentiate between various kinds of people, such as politicians, actors, or professors. Third, there are statistical machine learning approaches that have become increasingly popular. These approaches can broadly be divided into unsupervised and supervised machine learning. In unsupervised machine learning, the learning algorithm does not know the entity types and clusters the detected

[1] `http://areca.co`
[2] `http://palladian.ws`

mentions. This approach often relies on patterns and lexical resources such as WordNet [22]. Downey et al. [10] researched the task of detecting any type of entity in web texts instead of learning particular predefined classes. However, they do not propose a solution for entity classification, but rather only provide a means to delimit boundaries. The most common approach is supervised machine learning in which the NER is trained on a labeled corpus of text, e.g. "<PER>John Hiatt</PER> is a musician from <LOC>Indiana</LOC>". The NER builds a model from the training data and uses it to recognize the learned entity types for untagged texts later. Many machine learning techniques have been applied including hidden markov models [3], maximum entropy models [4], Support Vector Machines [2], decision trees [26], and conditional random fields [17]. Also, various researchers [30, 15, 16] have shown that combining several weak classifiers yields a stronger single classifier. Apart from these three main approaches, a number of hybrid methods have been developed which combine rules, lexicons, and machine learning, for example [19].

The problem of creating labeled training was quickly recognized and researchers have since tried to ease the compilation of training data. One method to train a NER is to use a list of seed entities per entity type. The contexts around the seeds can be gathered and used for learning rules. There were several experiments with the size of the seed lists varying from only 7 per entity type [9] up to 100 [5]. Collins and Singer [8] show that only as few as 7 seeds are needed to create well-performing rules. Cucerzan and Yarowsky [9] have tested the performance using between 40 and 100 seeds and found that the larger the number of seeds, the higher the F-Measure due to an increased recall. On the other hand, Buchholz and van den Bosch [5] showed that precision increased when using smaller lists with only a slight drop in recall. Seed lists are therefore an easy and fast way to train a NER, but as Meulder et al. [19] pointed out, the recall of those systems is usually quite low. The recall problem can be countered using the so-called "bootstrapping" technique, which means that the rules learned from the contexts of the seed entities are again used to find other entities which are then again seeds for the next learning phase [9, 24]. [27] use the web to overcome the sparseness problem with labeled data. In particular, they improve the best CoNLL 2003 NER F-Measure by more than 1% by using search engine hit counts to improve the boundary detection of the token-based NER. Furthermore, they use Hearst pattern [13] queries and WordNet with entity candidates to perform disambiguation between ambiguous entities.

Most approaches that we have reviewed research the problem on a very narrow domain of about four classes which are typically person, organization, location, and miscellaneous. Some papers introduce other types such as products [24], which is still very broad and insufficient for complex applications [11]. We want to be able to recognize a wide variety of different entity types without creating and maintaining a training corpus. Downey et al. [10] have realized this problem and researched the problem of "Web NER", that is, the problem of detecting entities of any type. While they perform only entity delimitation by finding the correct boundaries around the entities but not the entity type, we

want to scale the named entity detection and classification on the web level using a flat ontology of 22 entity types.

3 Palladian NER

Our hypothesis states that we can train a NER from the web using only seed lists for each type as learning input. In order to test our hypothesis, we designed a named entity recognizer that can be trained with sparsely-annotated training data. Usually, supervised NERs need to be trained on completely annotated input data such as the CoNLL 2003 dataset and each token must be labeled by an expert. In the case of the CoNLL corpus, this means that every token has either the label `PER` (person), `LOC` (location), `ORG` (organization), `MISC` (miscellaneous), or `O` (no entity / outside). As discussed in the previous section, compiling such a training set is tremendously labor intensive. Our NER must therefore be able to learn from sparsely-annotated texts in which not every token has a label. Thus, we have an open world instead of a closed world as in supervised learning. If a token has no label, we can not assume that it is not an entity (`O`).

3.1 Training the NER

We divide the training on sparse data into three parts: *creating the training data*, *training the text classifier*, and *analyzing the contexts* around the annotations.

Creating the Training Data The input for our named entity recognizer is not a completely annotated corpus but rather a set of *seed entities*. For example, for the concept Actor we could use "Jim Carrey" as one of the seeds. While there are no limitations on the choosing of the seeds, popular, and more importantly, unambiguous entities with many mentions on the web are the preferred input. Figure 3.1 shows the algorithm for creating the training data. We have n concepts $C = [S_1, ..., S_n]$, each with a set seeds S. We now query a search engine[3] with the exact seed name and its concept and take the top URLs from the result set. Each of the result URLs is downloaded and we extract the textual content of the page using the Palladian Content Extractor [29], removing the header, footer, and navigational elements to acquire only the "main content" of the web page. Next, we annotate all our seed entities for the given concept in the text and save it. We use XML annotations in the form of `<TYPE>seed</TYPE>`. The last operation is cleaning and merging the annotated text files to one. In the cleaning process, we remove all lines that have no annotated seed, are shorter than 80 character, or have no context around the seed.

Train the Text Classifier After generating the training data we train a dictionary-based text classifier using the seed entities and the surrounding con-

[3] We used `http://www.bing.com` in our experiments

```
begin
  mentions := 10
  for each concept in C
      S := seeds for concept
      for each seed in S
          URLs := querySearchEngine(seed,concept,mentions)
          for each url in URLs
              webPage := download(url)
              text := extractPageText(webPage)
              for each seed in S
                  text := annotateSeed(seed,text)
                  save(text)
              end
              cleanAndMerge()
          end
      end
  end
end
```

Fig. 1. Generating Training Data

texts that we found on the web. Unlike many other NERs, we do not employ a large feature vector of numeric features such as TFxIDF, token position, or length. Instead, we treat the entity classification purely as a text classification problem. The input for our text classifier is a sequence of characters that is then divided into character level n-grams. The dictionary is built by counting and normalizing the co-occurrences of one n-gram and an entity type. The dictionary might then resemble Table 1 in which each column is an entity type (Person, Location, and Product) and each row is a 7-gram. In each cell, we now have the learned relevance for each n-gram and entity type $relevance(ngram, entityType)$. The sum of the relevances in each row must add up to one. The n-gram "John" is more likely to indicate a Person ($relevance(r.John, Person) = 0.9$) than Location ($relevance(r.John, Location) = 0.05$) while the n-gram "Gibson" is a little bit more likely to be a Person than a Product ($relevance(sonjr., Person) = 0.5$.

n-gram	Person	Location	Product
r. John	0.9	0.05	0.05
Indiana	0.1	0.85	0.05
son jr.	0.5	0.1	0.4

Table 1. N-Gram dictionary with relevances for entity types

We build four dictionaries of this kind. One holds only on the seed names and their relevance with the entity types. A second one uses the complete context before and after the annotation within a certain window size. A third one uses only the three context words before and after the mention of the seed entity. A fourth one holds case signatures for all tokens in the training data. We store three case signatures, "A" for completely uppercase words, "Aa" for capitalized words, and "a" for lowercase words. We will show how we use these dictionaries to classify entity candidates in Section 3.2.

Analyze Contexts Apart from training a text classifier on the context around the seeds, we also build a dedicated context dictionary with context patterns that we find adjacent to the entities. The rationale behind this approach is that the sequence of words before or after a word are good indicators for the entity type. For example, consider the following phrases: "X traveled to Y", "X was born in Y", or "X came back from Y". All of these two to three word contexts indicate that Y might be a location when used as a left context pattern and that X might be a person when seen as a right context pattern. We therefore build a dictionary similar to the one shown in Table 1, but with context phrases instead of n-grams. This approach is similar to [11]. We use context phrases with a length between one and three words. Also, we map all numeric expressions to the word "NUM". This gives us a higher recall in context phrases with numbers. For instance, it does not matter whether the phrase is "X paid 2 dollars" or "X paid 3 dollars" so we capture "paid NUM dollars" as the context phrase. The context dictionary will be used in the last step of the entity recognition in Section 3.2.

3.2 Using the NER

Once we have trained our named entity recognizer on the automatically generated training data, it is ready to be used. We divide the recognition problem into three parts: *entity detection*, *entity classification*, and *post processing*. These steps are executed sequentially on each given text.

Entity Detection In the entity detection phase we need to find entity candidates in the text. An entity candidate is a sequence of characters of an unknown type. The output of the entity detection phase is a list of candidate annotations. For instance, given the text "John Hiatt is a great musician from Indiana, USA", we expect the entity detector to tag the candidates as "<CANDIDATE>John Hiatt</CANDIDATE> is a great musician from <CANDIDATE>Indiana</CANDIDATE>, <CANDIDATE>USA</CANDIDATE>". Our NER is supposed to work for English language texts only. We therefore use a rule-based detection with regular expressions to find possible entity mentions. In particular, our expression covers sequences of capitalized words, but also allows a few lower-cased words. For example it covers "of" in order to detect "United States of America". For each detected entity candidate, we perform the same feature generation as shown in Section 3.1.

Entity Classification The second phase is the classification of the candidates. To classify an entity candidate, we again create all n-grams ($3 <= n <= 7$), look up the relevance scores in the dictionary, and assign the entity type with the highest score S to the candidate. The score for each entity type and given entity candidate is calculated as shown in Equation 1 and 2 where $N_{Candidate}$ is the set of n-grams for the given entity candidate and T is the set of possible types that the NER was trained to recognize.

$$S(type|candidate) = \sum_{n \,\epsilon\, Ncandidate} relevance(n, type) \tag{1}$$

$$type = \arg \max_{type \,\epsilon\, T} S(type|candidate) \tag{2}$$

Post Processing In the last phase of entity recognition we filter the classified entities, change their boundaries if necessary, and reclassify their types by using the learned patterns. We apply the following post processing steps.

- We remove all entities that are date fragments such as "Monday" and "July". This step is necessary because these fragments are capitalized like the entities we actually want to detect, and because they fall under the tagging rules of the entity candidate detector. Of course, we lose a little recall since there are people named "April" and movies with the name "August", but the precision rises significantly more than the recall drops when applying this filter.
- We remove those date fragments from other entity candidates and change their boundaries. For instance, "July John Hiatt" becomes "John Hiatt".
- In the entity detection phase, we generate many entity candidates that consist of words at the beginning of a sentence. For instance with "This is a sentence", we would generate "This" as a candidate entity, when it is not. To filter out these incorrect detections, we employ the case dictionary that we have generated in the training phase. We calculate the ratio of uppercase to lowercase occurrences and remove the entity if the ratio is lower than or equal to one. The rationale behind this approach is that capitalized words at the beginning of sentences might appear more often in a lowercase form indicating that they are not entities. The word "This" from the example has a very low uppercase to lowercase ratio since it most often appears in the lowercase form. We borrow this idea from Millan et al. [20].
- We switch the classified entity type when we found the entity in the training data but classified it incorrectly.
- We now apply the information from the context dictionary that we trained. Again, we take the context around the candidate and look up the score of each entity type matching the context words. We merge the score outcomes of the entity classification, the context classification, and the context words to reassign the most likely entity type to the candidate. For example, if we have classified "Paris" as a person but now the context "born in Paris" suggests that the candidate is much more likely a location, the context classifiers overrule the candidate-only classifier and switch the label. We now use our

set D of dictionaries. These are (1) candidate-only, (2) context n-grams, and (3) context patterns. As shown in Equation 3 we combine the scores. We then use Equation 2 to get the most likely entity type for the candidate.

$$S(type|candidate) = \sum_{d \,\epsilon\, D} S_d(type|candidate) \tag{3}$$

– In the last step, we use the learned left context information to change the boundaries on multi-token candidates. For example, we have detected the candidate "President Obama", but knowing that the token "President" appeared many times before the actual entity in the training set, we drop this token and change the boundaries so that the candidate represents "Obama" only.

4 Evaluation

4.1 Datasets

We use two datasets for our evaluation, the CoNLL 2003 and the TUD 2011 dataset. In this section, we will describe the properties, similarities, and differences between the datasets.

CoNLL 2003 The CoNLL 2003 dataset[4] was developed for the shared task on CoNLL in 2003. The participants in that task were asked to provide a language-independent named entity recognition tool to work on English and German data recognizing the types: person, organization, location, and miscellaneous. In our evaluation we only use the English part of the dataset. The dataset also comes with additional data such as list of names, POS-tags, and non-annotated data which we do not use in our experiments. The data is separated into training (68%), validation (for tuning the systems, 17%), and test (15%). A Reuters corpus of news wire articles was annotated by the University of Antwerp for this dataset.

While the CoNLL 2003 dataset is a valuable resource it comes with a number of drawbacks. Most importantly, only three entity types in addition to `MISC` type are tagged. We have noticed that the `MISC` type very often represents a nationality identifier such as "Spanish". Since there are many more entities that could fall into this category (products, buildings, landmarks, etc.) it is questionable whether the type of articles covers enough variety. As we have argued in the motivation of this paper, this categorization is too broad for many real world applications. Therefore, the performance levels that NERs can reach on this dataset might be misleading. It is unclear whether they would work with a much finer grained classification. Furthermore, we found that some documents in the dataset do not seem to resemble a real text but rather look as if they were scraped table contents.

[4] `http://www.cnts.ua.ac.be/CoNLL2003/ner/`

TUD 2011 The TUD 2011 dataset[5] was developed at the Dresden University of Technology to evaluate Named Entity Recognition on a finer grained level. Furthermore, it was important to us that we did not get data solely from one source otherwise the writing tends to be too homogeneous. We therefore used different web pages as sources for the dataset. Two annotators spent about 100 hours annotating 22 entity types in the dataset. In the first step, 20 seed entities per entity type were collected from web sources. We then used the algorithm from Figure 3.1 to automatically find possible mentions of the seed entities on the web. Documents for each entity type were then hand tagged for about four hours. We tried to assign the most specific type to each entity; if there was no fitting type we went up in the hierarchy and if nothing matched there, we had to assign `MISC`. For example, "Jim Carrey" would be annotated as `Actor` while "Bill Gates" would be a `Person` since there is no specific matching type. The 127 documents are separated into training (50%) and test (50%) sets.

Table 2 shows the number of tagged entities per type in the training and test part of the dataset. While the five top level types are almost evenly distributed, the 17 more specific types show more variation in the number of annotated instances. This variation is due to the fact that country names, for example, also occur in documents about airplanes, but not the other way around. Altogether, 3,400 mentions were annotated.

Entity Type	Training	Test	Total	Entity Type	Training	Test	Total
Person	216	166	382	Car	19	14	33
Politician	54	66	120	Comp. Mouse	18	21	39
Actor	59	28	87	Movie	28	30	58
Athlete	60	78	138	Mobile Phone	24	24	48
Location	178	145	323	Organization	201	231	432
Country	143	119	262	Newspaper	70	39	109
City	142	142	284	Team	60	75	135
Lake	28	14	42	University	14	17	31
Airport	40	30	70	Restaurant	19	20	39
Product	64	44	108	Band	37	20	57
Airplane	92	118	210	Misc	243	150	393

Table 2. Number of tagged entities per type

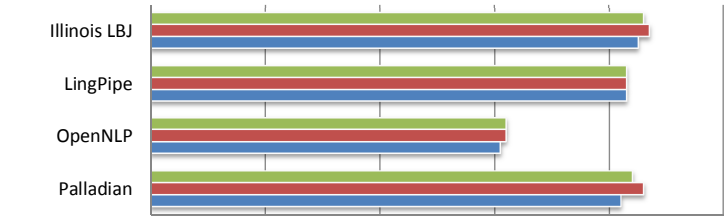
4.2 Comparisons

In this section, we evaluate our named entity recognizer on the two datasets and compare it with state-of-the-art NERs. In particular, we compare it to LingPipe[6] [1], which uses a character language model on top of a hidden markov

[5] `http://areca.co/7/Web-Named-Entity-Recognition-TUDCS4`
[6] `http://alias-i.com/lingpipe`

model, OpenNLP[7], which uses a maximum entropy approach for tagging, and the Illinois LBJ Tagger[8] [25], which is based on conditional random fields. All these recognizers work in a supervised manner, needing manually-labeled training data. Palladian works either in supervised mode (named just "Palladian" in the charts) but can learn from sparsely-annotated training data on the web, too (named "Palladian Web" in the charts). We use the MUC evaluation scores for precision, recall, and the F1 measure. Furthermore, it is important to note that all evaluation is done on "unseen" data, that is, we ignored annotations in the test set which could have been learned in the training data. We do this to avoid as much dictionary learning as possible since our goal is to extract new entities from text, not primarily to recognize what we know already.

Evaluation on CoNLL 2003 Figure 2 shows the comparison of the four NERs on the CoNLL data. All taggers were trained on 68% training data and tested on 15% test data from the dataset. This graphic shows what we can reach when using supervised learning with the training data.



	Palladian	OpenNLP	LingPipe	Illinois LBJ
MUC F1	0.84	0.62	0.83	0.86
MUC Recall	0.86	0.62	0.83	0.87
MUC Precision	0.82	0.61	0.83	0.85

Fig. 2. Comparison of recognizers with supervised learning on CoNLL

Figure 3 shows the performance of the NERs based on the size of the training data. We evaluated the performance by continuously increasing the training set by ten documents (in total, CoNLL provides about 600 training documents). We can see that all NERs benefit from more training data. The dashed line signifies the performance of Palladian Web – Palladian in web training mode. We used 1 to 50 seed entities per type and automatically generated training data using the algorithm from Figure 3.1. The upper x-axis shows the number of seeds that were used for training Palladian Web. While this approach works **without** hand-labeled training data, the F1-Score is about 29% below the maximum F1-Score

[7] `http://incubator.apache.org/opennlp/`
[8] `http://cogcomp.cs.illinois.edu/page/software_view/4`

of the best tagger when using completely annotated training data. Still even for 20 training documents the web training approach still outperforms LingPipe and up to over 250 training documents it beats OpenNLP's performance.

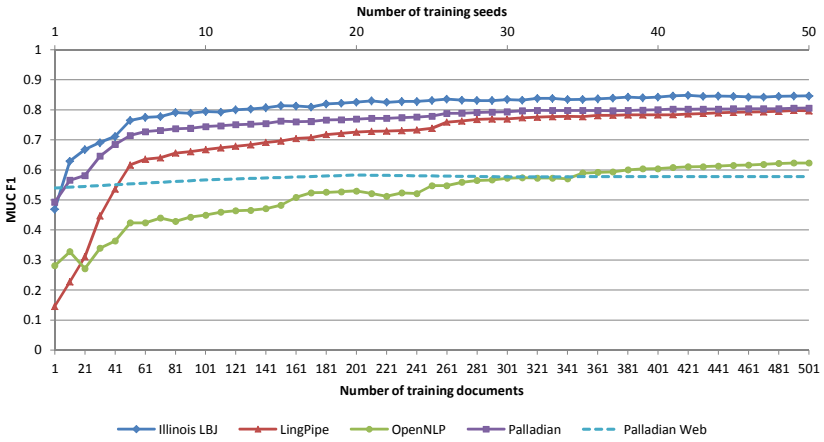
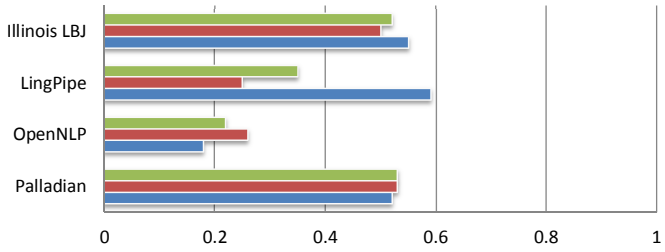


Fig. 3. Comparison of recognizers based on the size of training data on CoNLL

Evaluation on TUD 2011 Figure 4 shows the comparison of the four NERs on the TUD data. All taggers were trained on 50% training data and tested on the other 50% test data from the dataset. It is clear that the performance is dramatically worse for all the NERs compared to the CoNLL dataset. The worsened performance stems from a smaller number of training documents, more diverse training data, and more importantly the higher number of entity types that need to be recognized. From this chart, we can conclude that even state-of-the-art NERs perform poorly on diverse web text when the task is to extract unseen entity mentions. The performance could be improved by allowing gazetteers, but this is not our focus.

Figure 5 shows the performance of the NERs based on the size of the training data from the TUD dataset (in total, the dataset provides about 60 training documents). We increased the number of training documents by five for each evaluation. We can see a similar behavior of the taggers as in Figure 3. We trained the Palladian Web tagger the same way as before, supplying it with automatically generated training data. It takes the Illinois LBJ tagger about ten training documents to reach the performance of the Palladian Web tagger trained on 10 seeds. This time, however, the number of training documents is too small for LingPipe and OpenNLP to reach the automatically trained tagger's performance at all. Training Palladian Web with 50 seed entities per type performs only about 9% worse than the Illinois LBJ after 56 training documents. We can see a promising direction of automatically training the NER on the web



	Palladian	OpenNLP	LingPipe	Illinois LBJ
MUC F1	0.53	0.22	0.35	0.52
MUC Recall	0.53	0.26	0.25	0.50
MUC Precision	0.52	0.18	0.59	0.55

Fig. 4. Comparison of recognizers with supervised learning on TUD

since acquiring training data is extremely costly, and when a new entity type should be extracted, the complete data needs to be manually re-annotated.

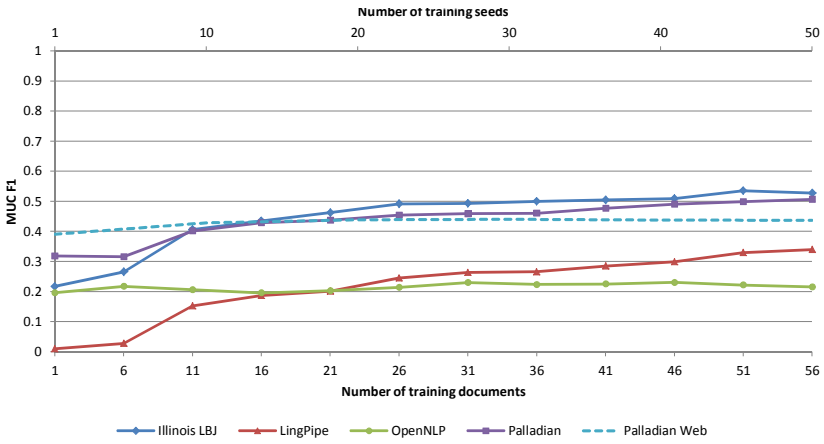


Fig. 5. Comparison of recognizers based on the size of training data on TUD

5 Conclusion & Open Questions

In this paper, we have presented the named entity recognizer Palladian, which can be automatically trained on the web requiring only a small number of seed entities per entity type. While this approach is outperformed by training supervised state-of-the-art NERs on manually-labeled training data, it is rather competitive when little training data is given and many different entity types

should be extracted. Furthermore, we have shown that named entity extraction which focuses on recognizing unknown entities is a rather difficult problem when the task is done on heterogeneous web texts and a larger number of entity types need be recognized. In our future studies, we will combine the presented NER with the entity extraction techniques from [28] to improve the quantity and quality of the entity extractions.

References

[1] Alias-i. Lingpipe 4.0.1, 2011. `http://alias-i.com/lingpipe`.

[2] M. Asahara and Y. Matsumoto. Japanese named entity extraction with redundant morphological analysis. In *Proceedings of Human Language Technology Conference (HLT-NAACL)*, pages 8–15, 2003.

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

[4] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. NYU: Description of the MENE named entity system as used in MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, volume 6, 1998.

[5] S. Buchholz and A. van den Bosch. Integrating seed names and n-grams for a named entity list and classifier. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, pages 1215–1221, 2000.

[6] L. Chiticariu, R. Krishnamurthy, Y. Li, F. Reiss, and S. Vaithyanathan. Domain adaptation of rule-based annotators for named-entity recognition tasks. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1002–1012, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL `http://portal.acm.org/citation.cfm?id=1870658.1870756`.

[7] W. W. Cohen. Fast effective rule induction. In *Machine Learning, International Workshop then conference*, pages 115–123. Morgan Kaufmann Publishers, 1995.

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

[9] S. Cucerzan and D. Yarowsky. Language Independent Named Entity Recognition Combining Morphological and Contextual Evidence. In *Proceedings of the Workshop on Very Large Corpora at the Conference on Empirical Methods in NLP.*, pages 90–99, 1999.

[10] D. Downey, M. Broadhead, and O. Etzioni. Locating complex named entities in web text. In *Proceedings of IJCAI*, 2007.

[11] M. Fleischman and E. Hovy. Fine Grained Classification of Named Entities. In *Proceedings of the 19th international conference on Computational linguistics*, volume 1, pages 1–7. Association for Computational Linguistics, 2002.

[12] R. Grishman and B. Sundheim. Message understanding conference-6: A brief history. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 466–471. Association for Computational Linguistics Morristown, NJ, USA, 1996.

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

[14] F. Iacobelli, N. Nichols, and L. B. K. Hammond. Finding new information via robust entity detection. In *Proactive Assistant Agents (PAA2010) AAAI 2010 Fall Symposium*, 2010.

[15] D. Klein, J. Smarr, H. Nguyen, and C. D. Manning. Named entity recognition with character-level models. In *Proceedings of CoNLL*, volume 3, 2003.

[16] Z. Kozareva, B. Bonev, and A. Montoyo. Self-training and co-training applied to spanish named entity recognition. *MICAI 2005: Advances in Artificial Intelligence*, pages 770–779, 2005.

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

[18] D. D. McDonald. Internal and external evidence in the identification and semantic categorization of proper names. *Corpus processing for lexical acquisition*, pages 21–39, 1996.

[19] F. D. Meulder, W. Daelemans, and V. Hoste. A named entity recognition system for dutch. *Language and Computers*, 45(1):77–88, 2002. ISSN 0921-5034.

[20] M. Millan, D. Sánchez, and A. Moreno. Unsupervised Web-based Automatic Annotation. In *Proceeding of the 2008 conference on STAIRS 2008: Proceedings of the Fourth Starting AI Researchers' Symposium*, pages 118–129. IOS Press, 2008.

[21] D. Milne and I. H. Witten. Learning to link with wikipedia. In *Proceeding of the 17th ACM conference on Information and knowledge management*, pages 509–518. ACM, 2008.

[22] D. Nadeau and S. Sekine. A Survey of Named Entity Recognition and Classification. *Named Entities: Recognition, Classification and Use*, pages 3–28, 2009.

[23] D. Nadeau, P. D. Turney, and S. Matwin. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. In *Proceedings of the Canadian Conference on Artificial Intelligence*, pages 266–277. Springer, 2006.

[24] C. Niu, W. Li, J. Ding, and R. K. Srihari. Bootstrapping for named entity tagging using concept-based seeds. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: companion volume of the Proceedings of HLT-NAACL 2003–short papers-Volume 2*, pages 73–75. Association for Computational Linguistics, 2003.

[25] L. Ratinov and D. Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics, 2009.

[26] S. Sekine. NYU: Description of the Japanese NE System used for MET-2. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998.

[27] G. Szarvas, R. Farkas, and R. Ormándi. Improving a state-of-the-art named entity recognition system using the world wide web. *Advances in Data Mining. Theoretical Aspects and Applications*, pages 163–172, 2007.

[28] D. Urbansky, M. Feldmann, J. A. Thom, and A. Schill. Entity Extraction from the Web with WebKnox. In *Proceedings of the Sixth Atlantic Web Intelligence Conference*, 2009.

[29] D. Urbansky, K. Muthmann, P. Katz, and S. Reichert. Palladian: A toolkit for Internet Information Retrieval and Extraction. Website, May 2011. `http://www.palladian.ws/documents/palladianBook.pdf`.

[30] D. Wu, G. Ngai, M. Carpuat, J. Larsen, and Y. Yang. Boosting for named entity recognition. 20:1–4, 2002.