Automatic gazette creation for named entity recognition and application to resume processing

Conference Paper · January 2012

DOI: 10.1145/2459118.2459133

3 authors:

Sachin Pawar
Tata Consultancy Services Limited
44 PUBLICATIONS **183** CITATIONS
SEE PROFILE

Rajiv Srivastava
Tata Research Development and Design Centre
17 PUBLICATIONS **60** CITATIONS
SEE PROFILE

Girish Palshikar
Tata Consultancy Services Limited
156 PUBLICATIONS **1,145** CITATIONS
SEE PROFILE

Automatic Gazette Creation for Named Entity Recognition and Application to Resume Processing

Sachin Pawar, Rajiv Srivastava and Girish Keshav Palshikar
Tata Research Development and Design Centre,
54 B Hadapsar Industrial Estate, Pune 411013, India.

ABSTRACT

Named entities are important content-carrying units within documents. Consequently named entity recognition (NER) is an important part of information extraction. One fast and accurate approach to NER uses a *list* or *gazette* consisting of known instances. Gazette creation problem considers how to automatically create a comprehensive gazette from given unlabeled document repository. We describe an unsupervised algorithm for automatic gazette creation, which is modified from [5]. We propose a fast NER algorithm using large gazette and show that it significantly outperforms a naïve approach based on regular expressions. We describe experimental results obtained by using the system for gazette creation for various resume related named entities (e.g., ORG, DEGREE, EDUCATIONAL_INSTITUTE, DESIGNATION) and the associated NER on a large set of real-life resumes.

Categories and Subject Descriptors: I.3.7 [**Machine Learning**]: Text mining;

General Terms: Named Entity Recognition, Named Entity Extraction, Gazette Creation, Information Extraction, Information Retrieval.

Additional Key Words and Phrases: Resume processing.

Authors' addresses: {sachin7.p, rajiv.srivastava, gk.palshikar}@tcs.com.

DOI 10.1145/1559755.1559763
http://doi.acm.org/10.1145/1559755.1559763

1. INTRODUCTION

Named entities, both generic (e.g., names of persons, organizations, locations, dates, email addresses) as well as domain-specific (e.g., genes, proteins, enzymes, cells, organs, diseases) are important content-carrying units within most documents. The task of *named entity recognition* (*NER*) consists of identifying all occurrences of a given named entity type (e.g., ORG) in the given document. NER is a complex task due to various factors such as the noisy nature of documents, and language-dependent and semantics-based aspects of the task. Consequently, a large number of approaches have been designed for performing NER; see [4] for a survey. One simple approach to NER is to use a *list* or *gazette*; e.g., a gazette for the named entity DEGREE would consist of all known educational degree names, such as **`Bachelor of Arts, B.A., Ph.D.`** etc. The gazette-based approach results in fast and high-precision named entity recognition, since one simply looks for occurrences of any entries in the gazette – though occasionally one needs some post-processing to distinguish between an occurrence of **`London`** as CITY or as PERSON (e.g., **`Jack London`**). But the accuracy (recall) of the gazette-based approach is critically dependent on the completeness of the gazette used. Creating the gazette manually is effort-intensive, error-prone and subjective, since there may be thousands (or even millions) of examples for the named entity. One possible solution is to create the gazette (for a particular named entity) automatically from the large unlabeled corpus of text documents available either within the organization or over the Internet.

The problem then is how to automatically create a comprehensive gazette (for a particular named entity) with less effort, in less time and with high accuracy using a given document repository. In many organizations, large repositories of documents are available – e.g., scientific papers, resumes, news stories or insurance claims – which contain a large number of examples of the named entity of interest, though these examples are unlabeled (not marked as such). Hence such a document repository can be potentially useful as input to create a large gazette using supervised, semi-supervised or unsupervised *named entity extraction* (*NEX*) techniques.

Note that the NEX problem (i.e., named entity gazette creation) that we are talking about is related to but distinct from the NER problem. There is often a need for cleaning, post-processing and standardizing the named entity instances in an automatically created gazette. After creating and post-processing a large gazette, the problem of using the gazette for performing NER (finding all occurrences of any strings in the gazette in the given document), requires an efficient search and match algorithm.

In this paper, we propose a scalable, largely automated system for: (i) un-supervised creation of a gazette for any named entity; (ii) cleaning, post-processing and standardizing the named entity instances in an automatically created gazette; and (iii) an efficient search and match algorithm for gazette-based NER. We illustrate the use of this system in the HR domain. Specifically, we demonstrate how the system is used for automatically creating large gazettes for various named entities commonly found in resumes; e.g., DEGREE, EDUCATIONAL_INSTITUTE, ORG (employer or client), DESIGNATION etc. We also discuss how this system is used to post-process the automatically created gazettes.

This paper presents a domain-independent unsupervised gazette creation algorithm that works for many types of named entities. The input to the algorithm is a small set of *seed examples* (or *instances*) of the named entity of interest; and a set of text documents. The output is a gazette, which is a set of additional (new) instances of the named entity. Our algorithm is modification of the one described in [5], named BASILISK. Our specific contributions are as follows. (i) we have used our own set of features, many (not all) of which are tailored to the resume processing application we had in mind; (ii) we have added a significant variation to the original algorithm in the form of *negative features*, which are used to filter out incorrect instances of the named entity; and (iii) we have further tuned the algorithm (e.g., we have experimented with various new *g* functions – explained later) and (iv) demonstrated the use of this algorithm in a specific practical application: in creating gazettes for named entities that frequently occur in resumes. We have also made several other smaller changes to the original algorithm (like *feature partitioning* – explained later).

We present a fast algorithm to identify occurrences of the named entities in a given gazette in given resume i.e., to perform the gazette-based NER. The key contribution here is the idea to identify the "most important" words in a gazette entry and use them for indexing. The NER algorithm also detects "approximate" occurrences that "closely" match a gazette entry, without too much performance degradation. The sub-systems NEX and NER are part of the information extraction module 'ResumeExtractor', within a larger system 'ResumeCentre' that we are building for resume processing.

The paper is organized as follows Section 2 describes some related work. Section 3 describes the unsupervised algorithm for automatic gazette creation. Section 4 describes the algorithm for gazette-based NER. Section 5 describes experimental results obtained by using the system for gazette creation for various resume related named entities and associated NER on a set of real-life resumes. Section 6 discusses our conclusions and proposes some further work.

2. RELATED WORK

Supervised ML techniques attract the most attention of NER researchers because of their advantages such as high performance and domain neutrality (ease of adapting to different domains). Major disadvantage of such supervised techniques is that, they require a large correctly tagged corpus for training.

The unsupervised learning approaches reduce the need for a large correctly tagged corpus. [1] propose DL-CoTrain algorithm, which starts with a few simple seed rules (decision lists), identifies the entities in the corpus using these rules, exploits redundancy in the text (two features hint at the same entity) to identify new rules and then continues the cycle. It also proposes CoBoost, a boosting based unsupervised algorithm for NER. [5] proposes a bootstrapping algorithm for NEX. [2] presents a system called KNOWITALL, which implements an unsupervised domain-independent, bootstrapping approach to generate large facts of a specified entity (such as City or Film) from the Web. It starts with a few seed patterns – e.g., *NP such as NPList* – and uses them to deduce from the text cities such as London and Beijing that London and Beijing are cities. It contains (i) an algorithm to identify new extraction rules and (ii) a classifier based on point-wise mutual information for validating the extracted entities. [3] propose an approach where they first use an unsupervised approach similar to [1] and [2] to generate a gazette of instances of the entities (e.g., cities) and then use some effective heuristics to identify new instances of the entity.

3. UNSUPERVISED GAZETTE CREATION

3.1 Document Pre-processing

There is a need to perform certain cleanup and other pre-processing operations on the documents in the given corpus, before they can be given as input to the unsupervised gazette creation algorithm. The pre-processing includes spelling corrections, format conversion (e.g., Microsoft Word to plain text), adding markers to preserve information (e.g., bold, italics, underline, font size), sentence boundary detection, expansion of abbreviations such as Co. to Company, Ltd. to Limited, removing unwanted symbols/words/characters (e.g., replacing I.B.M. with IBM or replacing !!! with !) etc. Each of these pre-processing steps is implemented as a regular expression.

Table 1. Features used for unsupervised gazette creation.

Example Sentences	***Work Experience:*** *I have worked with Tata Consultancy Services from June 2009 to June 2011.* *Designation : Senior Software Engineer*	
Feature Description	**Feature Instance**	**Applied to Entity**
First Word	FW_Tata	Tata Consultancy Services
	FW_Senior	Senior Software Engineer
Last Word	Services_LW	Tata Consultancy Services
	Engineer_LW	Senior Software Engineer
First 2 Words	FW_Tata_Consultancy	Tata Consultancy Services
	FW_Senior_Software	Senior Software Engineer
Last 2 Words	Consultancy_Services_LW	Tata Consultancy Services
	Software_Engineer_LW	Senior Software Engineer
Context of previous and next word	Context_with_from	Tata Consultancy Services
	Context_:_NEWLINE	Senior Software Engineer
Word just before colon in the current line	CLW_Designation	Senior Software Engineer
Duration present in the current line	Context_Duration	Tata Consultancy Services
Duration present in the next line	Context_Duration_next	Work Experience
Duration present in the previous line	Context_Duration_prev	Senior Software Engineer
Last word of a section header which occurred within last 4 lines	SECT_Experience	Tata Consultancy Services, Senior Software Engineer

3.2 Gazette Creation Algorithm

The gazette creation algorithm mainly uses context-based features i.e., words that occur just before or after known instances of the named entity. The gazette creation process is unsupervised and requires minimal human intervention as the only manual input required is a few seed entries of the type we are interested in. For example, to create a gazette of ORG named entity from a corpus of resumes, we give a small list (say 10 entries) of known organization names such as **`Tata Consultancy Services`**, **`Tata Motors`** and **`Tech Mahindra Limited`**. The algorithm itself is independent of type of the named entity to be extracted. To create a gazette for a different type of named entity (e.g., DESIGNATION), only the seeds need to be replaced by the relevant instances.

Each feature is implemented using a simple *rule*. Such a rule is applied to a suitable text fragment (e.g., the sentence) containing a given instance of a known named entity and rule then extracts the value of the feature by performing an analysis of this text fragment. For example, the rule for the feature *CONTEXT_PREV_WORD_NEXT_WORD* extracts the words just before and just after the occurrence of a known named entity instance. Applying this rule for this feature to the named entity instance **`Tata Consultancy Services`** and the text fragment (sentence) **`I am currently working with Tata Consultancy Services since June 2009`**, the feature value extracted is "CONTEXT_with_since". We need a comprehensive set of features (and associated rules) to help in accurate gazette preparation. Many features are *generic* in the sense that they are useful for all types of named entities and are not specific to resume documents. *Specific features*, on the other hand, are tailored for resume documents. Table 1 shows the features that we have used; first 5 features are generic and the remaining are oriented towards resume documents. The detailed algorithm for gazette creation is as follows. Basically, the algorithm identifies a list of word-sequences, each of which is a potential occurrence of the named entity. Then it computes all features for all these instances which are seeds or not seeds (function get_feature_counts), and then ranks the features in terms of their importance, selecting top c of them for this iteration. Then it computes the final score for each non-seed instance, using the importance of the features which are applicable to that instance. Then it selects top e of these non-seed instances ranked with their final scores. These e instances are added to the seeds and the next iteration begins.

The algorithm terminates when the specified number of iterations is over or when no new instances of the named entity are found; we are also experimenting with some other termination conditions, including human monitoring of the results being produced, that are not discussed here. Each potential named entity instance essentially consists of a sequence of capitalized words, ignoring some words such as **`of`** or **`and`** that may occur between two capitalized words. The importance of each feature (function g) depends on its frequencies of occurrences for both seed and non-seed instances of named entities. If a feature is seen more frequently with seed instances but less frequently seen with non-seed instances then it is more important.

We have added *negative features* to the algorithm as an effective way to incorporate domain knowledge in the algorithm; e.g., DESIGNATION and ROLE are overlapping in the sense that the same string (e.g., **`Analyst`**) can occur either as ROLE or as DESIGNATION. Hence, when creating a gazette for DESIGNATION, we need to avoid considering features that are more relevant for ROLE. Such ROLE-oriented features can be included in the list of negative features, so that entity instances which have those features will not be added to the seeds list, thereby improving the overall accuracy of the algorithm. For example, suppose we have seen a positive example text **`Designation: Analyst`**, and hence **`Analyst`** gets added to the seeds. In the next iteration, upon seeing the text **`ROLE: Analyst`**, the algorithm may consider feature CLW_Role (which detects patterns such as text : known-seed – here text is set to **`Role`**), which leads the algorithm astray; now it starts recognizing incorrect instances for DESIGNATION like **`testing`** from text such as **`Role : testing`**. This can be avoided by making the feature CLW_Role as a negative feature.

We experimented with some new g functions than that used in the original BASILISK algorithm. To determine the score for each feature, original algorithm uses the formula, $(D_2/ D_1) * \log(D_2)$, where D_1 is the number of *distinct* entity instances where the feature is present and D_2 is number of *distinct seed* entity instances where the feature is present. We observed that, when we start with a relatively small initial seeds list (< 10), instead of considering the distinct counts in the above formula, if we consider the *total* (rather than distinct) number of occurrences in the whole corpus, the feature scores in the initial iterations are much more reliable. Some g functions that empirically performed better in our case are as follows:

$g(C_1, C_2) = (C_2/ C_1) * \log(C_2)$
$g(C_1, C_2, D_2) = (C_2/ C_1) * \log(C_2) * D_2$

where C_2 is the total number of occurrences of seed entity instances in the corpus where the feature is present and C_1 is the total number of occurrences of entity instances where the feature is present.

The other change that we made to the original algorithm, which we experimented with but haven't included in the detailed algorithm due to space restrictions, is *feature partitioning*. We observed that for some complex entity types, it is better to view the feature rules as the union of two mutually exclusive partitions, which are different in some natural way. For example, one partition can be of only context features like CONTEXT_PREV_WORD_NEXT_WORD and CONTEXT_DURATION and other partition can contain all the remaining features. We then impose constraints like: the entity instance should be extracted by at least one feature from each partition. Such constraints are helpful for the entity types where more diverse evidence is necessary.

```
algorithm BASILISK_MODIFIED
input  D = {$D_1, D_2, …, D_I$} // set of I documents
input  S = {$S_1, S_2, …, S_J$} // set of J seed instances for the named
                                 // entity of interest
input  F = {$F_1, F_2, …, F_K$} // set of K feature rules
input  N = {$N_1, N_2, …, N_L$} // set of L negative features
input  m // Maximum number of iterations
input  e // No. of entity instances that must be added to the seed after
         // every iteration
input  c // Top c features to be considered in each iteration
output S // set of seed entries grown over multiple iterations
$L_1$ = get_feature_counts(D, F, S, FALSE);
i = 0; // iteration number
while $i < m$ do
    $L_2$ = get_feature_counts(D, F, S, TRUE);
    // selection of top features
    $L = \emptyset$; // sorted list of feature instances based on score
    for each featureInstance in $L_2$ do
        $c_1 = L_1$[featureInstance]; // count of featureInstance in $L_1$
        $c_2 = L_2$[featureInstance]; // count of featureInstance in $L_2$
        s = g(c_1, c_2); // e.g., c_2/c_1 or (c_2/c_1)log(c_2)
        Add (featureInstance, s) to L;
    end for
    L = select top c features from L; // keep only top c features in L
    // Selection of entities
    E = ∅; // Sorted list of entity instances based on score
    for each document d in D do
        P = all phrases in d which are potential named entity instances;
        for each p in P do
            count = 0; // no. of features applicable to p
            if p ∈ S then continue; end if; // add only new instances
            isNegFeaturePresent = FALSE;
            scoreOfP = 0;
            for each f in F do
                featureInstance = apply f on p ;
                if featureInstance ∈ N then
                    isNegFeaturePresent = TRUE; break;
                end if
                if featureInstance ∈ L then
                    scoreOfP += L[featureInstance]; count++;
                end if
            end for
            if isNegFeaturePresent == TRUE then break; endif
            if count > 0 then Add (p, scoreOfP/count) to E; endif
        end for
    end for
    Select top e entities from E and add them to S; i++;
end while
function get_feature_counts(D, F, S, flag)
LX = ∅; // hash table mapping feature instance to count
for each document d in D do
        P = all phrases in d which are potential named entity instances;
        for each p in P do
            for each f in F do
                featureInstance = apply f on p;
                if flag == FALSE then
                    Add featureInstance to LX and increment the count;
                else if p ∈ S then // do only if the instance is a seed
                    Add featureInstance to LX and increment the count;
                endif
            end for
        end for
end for
return(LX);
```

3.3 Gazette Post-processing

After running the algorithm, a gazette is created that consists of many new instances of the named entity of interest. A number of post-processing steps are applied to this gazette to "clean it up", so that it is ready to be deployed for NER. For instance, some instances may include an extra word either at the beginning or at the end, which is removed. The *de-duplication* step consists of identifying (but *not* removing) many similar instances that differ in minor ways such as capitalization or a period, minor spelling variations, word order changes or inclusion of a superfluous word such as **`the`**. Sometimes one instance is a short-form or an abbreviation of another (e.g., **`Master of Arts`** and **`M.A.`**). The standardization step consists of creating a mapping from an instance to a *standard representation* of that instance, so that similar instances are mapped to the same standard instance. The standard representation is useful for post-NER activities. For example, the same ORG named entity can occur as several instances:

```
Welingkar Institute of Engineering and Management
Welingkar Institute of Management and Engineering
The Wellingker Inst of Engg. & Mgmnt.
```

Detecting whether two instances are "highly similar" and hence belong to the same instance of the named entity is the critical step in the standardization process. Essentially, two instances (word sequences) X and Y are highly similar if their similarity score is above a specified threshold. The similarity score is computed after both X and Y are "cleaned up". The similarity score can be computed based on various types of similarity such as: (i) X is an abbreviation of Y or vice versa; or (ii) X contains the same words as Y except in possibly different order; or (iii) X and Y are "highly similar" in terms of edit (or Levenshtein) distance. The string clean up step consists of removing stop-words (such as **`of`**, **`the`**), expanding standard short-forms (e.g., **`Inst.`** to **`Institute`**) etc. Each set of similar instances (all of which are similar to each other) is then mapped to a standard representation, which might be user-specified or say the longest or the most frequent instance in that set. After all the automated post-processing is completed, the user can still manually edit and clean-up the created gazette, if required.

4. GAZETTE-BASED NER

After creating the gazette G, the next task (NER) is to apply the gazette G to a given document D and identify all occurrences in D of all entity instances of G. A naïve algorithm for this task is as follows and simply looks for all occurrences of each entry of G in D:

```
algorithm NAÏVE_GAZETTE_BASED_NER
input   D // document
input   G // gazette
output  S // set of gazette entries present in document D
S = ∅; // initially empty
for each instance x ∈ G do
    Search x in D and add to S all occurrence of x in D; // use regex
end for
return(S);
```

The search step can be carried out using, say, regular expression. This approach is cumbersome if the gazette G contains lots of entries (say, 10,000). In such cases, we need a more efficient algorithm. We propose a gazette-based NER algorithm which leverages importance of words in the named entity instances in G for efficient search. The function createIndex is called once after reading in the given gazette (from a file, say) and uses the most important words of each named entity instance in G for preparing n-level indexes for quick search.

```
// index each gazette entry based on n most important words in it
function createIndex(G, n)
TRIE t[n]; // t[k] = trie for storing k^th important word in all entries
for each instance e ∈ G do
    WL = list of words in e; // split gazette entity on word boundary
    Sort WL on importance of words in it; // e.g. based on IDF
    for i = 1 to n do
        insert WL[i] in t[i] along with reference to e;
    end for
end for
```

The algorithm QUICK_SEARCH uses the indexes to search for all occurrences in the given document of each entry in the given gazette. The indexes are assumed to be available as global TRIE data

Table 2. Examples entries in automatically created gazettes.

ORG	DEGREE	EDUCATIONAL_INSTITUTE	DESIGNATION
HCL Technologies BPO	B.A.	Bengal Engineering and Science University	Programmer Analyst
ACCENTURE Services Mumbai	B.B.A.	Indian Institute Of Mass Communication	Sr Recruiter
Cognizant Technology Solutions	Bachelor of Applied Science	Univ. of Pune	Technical Support Executive
TCS Delhi	Bachelor of Arts	University of Southern Queensland	Sr. Finance Executive
CGI Netvorks	Bachelor of Business Studies	Govt. Autonomous Model Science College	Senior Business Analyst
Patni Computer Services	Bachelor of Commerce	RYK Science College	Technical Project Manager
Paxar Corporation	Bachelor of Computer Aplications	Shivaji Science College	Relationship Officer
Vision Infotech	Diploma of Engineering	Institute of Cost and Works Accountant of India	Jr. Floor Supervisor
GlobalCynex Inc	Master of Foreign Trade	SRM School of Management	Software Developer
IQ Technologies LLC	Diploma of Computer Engineering	Government College Of Engg	Sr. Faculty

structures. As an example, consider the gazette of EDUCATIONAL_INSTITUTE, which has around 6400 entries. Consider an entry **`Indian Institute of Technology Kharagpur`**. This entry is first split into constituent words and importance of each word is determined using *inverse document frequency* (*IDF*); words occurring in fewer entries are more important. Here, **`Kharagpur`** turns out to be most important as it has the least IDF when compared to **`Indian`**, **`Institute`** and **`Technology`**. Therefore, the most important word for this entry is **`Kharagpur`**. Now suppose, a line in resume is as follows: **`Graduated from Indian Institute of Technology Kharagpur in 2002`**. Each of the word in this sentence is looked up in the index to find the possible matches. Finding **`Kharagpur`** narrows down the search to only 2 entries: **`VINOD GUPTA SCHOOL OF MGMT, KHARAGPUR`** and **`Indian Institute of Technology Kharagpur.`** At this point, the string similarity of the 30-character context on both sides of the word **`Kharagpur`** and each of the above entries is calculated. We get above-threshold match for the later entry, since the context and the entry both include words **`Indian Institute of Technology`**.

```
algorithm QUICK_SEARCH
input  D // document
input  G // gazette
input  n; // number of indexes to be used
output S // set of gazette entries present in document D
for each sentence s in D do
    WL = list of words in s; // split sentence on word boundary
    for each word w in WL do
        L = Look up w in top n indexes to get list of gazette entries;
        matchFound = FALSE;
        for each entry x in L do
            Let y = suitable length context around w in s;
            a = similarity(x, y);
            if (a > threshold) then
                add x to S; matchFound = TRUE; break;
            end if
        end for
        if matchFound == TRUE then break; end if
    end for
end for
```

5. EXPERIMENTAL RESULTS

5.1 Evaluation of Gazette Creation Algorithm

We tested both the BASILISK_MODIFIED algorithm for automatic gazette creation and QUICK_SEARCH algorithm for gazette-based NER on a set of 4000 resumes. We ran the BASILISK_MODIFIED algorithm on these 4000 resumes to create the gazettes for 4 types of named entities: ORG, EDUCATIONAL_INSTITUTE, DESIGNATION and DEGREE. Table 2 shows some example entries from these 4 automatically created gazettes.

We used 3-5 seed examples in each case (Table 3) and the maximum number of iterations was set to $m = 20$. We used $e = 50$ so that at most 50 new entities were added to the set of seeds in each iteration. Further, we restricted the gazettes to have at most 1000 entries. We manually checked and validated each entry in each gazette to calculate the precision (i.e., percentage of correct entries). In these experiments, we did not focus on computing recall i.e., how many instances of the named entity which were present in the resumes were detected and added to the gazette. One reason is that the 'ResumeExtractor' also has non-gazette-based extraction rules for these 4 types of named entities, which "catch" any instances which were not included in the gazette. Also, we felt that at this stage it is more important to focus on the quality (rather than the coverage) of the named entity instances discovered by the automatic gazette creation algorithm. Note the high quality discoveries (very different from seeds) made by the automatic gazette creation algorithm; e.g., **`Diploma of Engineering`**, **`Vision Infotech`** and **`Sr Recruiter`**.

Table 3. Seed instances used for gazette creation.

ORG	DESIGNATION
Infosys Technologies Ltd	Business Analyst
Nextel Communications Inc	Consultant
Tech Mahindra Limited	Software Engineer
EDUCATIONAL_INSTITUTE	**DEGREE**
Bengal Engineering College	B.E.
Indian Institute of Technology	Bachelor of Engineering
Institute of Management Technology	M.B.A.
U.P. Technical University	M.C.A.
University of Pune	Master of Computer Application

The precision for the above 4 types of named entities is shown in Table 4. Precision is a bit low for DEGREE because some people include specialization as part of the degree (e.g., **`B.E. Computer Science`**); these extraneous words lead the algorithm astray.

Table 4. Precision for automatically created gazettes.

Entity Type	No. of Entities Discovered	No. of Iterations	Precision
ORG	1000	20	93.1%
EDUCATIONAL_INSTITUTE	881	19	95.68%
DESIGNATION	1000	20	82.5%
DEGREE	217	5	83.87%

The same features were used for all 4 types of named entities. For EDUCATIONAL_INSTITUTE and DEGREE, the algorithm stopped because there were no more new instances to extract, which were matching the top features. For ORG and DESIGNATION, the algorithm stopped because the number of iterations reached the maximum limit set (20). While Table 4 shows the final and overall precision for the entire gazette, Figure 1 shows how the error rate (= 100 – precision) varied across individual iterations for each of the 4 types of named entities. For DEGREE, the error rate increased sharply with iteration number, primarily because there are only a few instances for DEGREE, which the algorithm discovers quickly in the first few iterations. For other 3 types of named entities, the error rate keeps fluctuating, but we have no clear explanation for why that is the case. We have observed that the error rate always shows an increasing trend (or stays high) when the gazette is more or less complete.

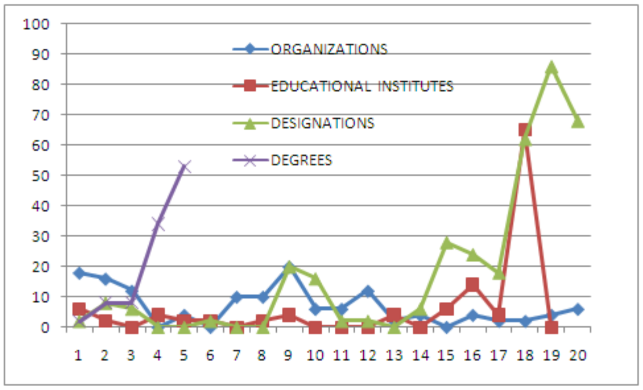

Figure 1. Variation of the error rate across iterations.

5.2 Evaluation of Gazette-based NER Algorithm

We tested the QUICK_SEARCH algorithm for gazette-based NER on a set of 4000 resumes and a list of 6400 entries for EDUCATIONAL_INSTITUTE. This gazette was created by running the BASILISK_MODIFIED algorithm on large set of resumes. For exact matching, the algorithm performed much better than the naïve matching algorithm based on regular expressions. For exact matching, using only 1 TRIE, our algorithm took 23 seconds, whereas regular expression based naïve algorithm took 1200 seconds. In case of approximate matching, our algorithm took 380 seconds which is still almost 3 times better than the naïve algorithm. As shown in Fig. 2, QUICK_SEARCH algorithm significantly outperforms the naïve regular expression based NER algorithm. The increase in time taken for NER by our algorithm is much smaller than the naïve algorithm as the gazette size increases.

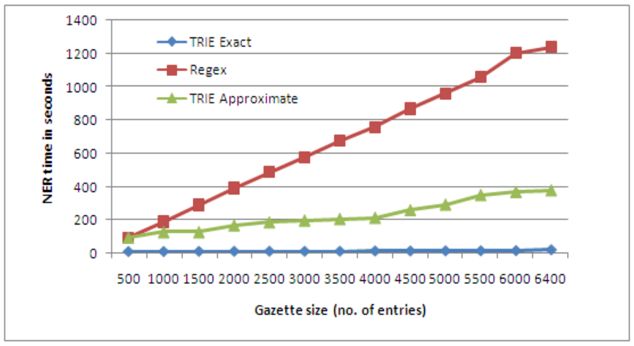

Figure 2. Performance of gazette-based NER algorithm.

6. CONCLUSIONS AND FURTHER WORK

In this paper we proposed a modified version of the unsupervised automatic gazette creation algorithm in [5]. We also proposed a fast NER algorithm using large gazette and showed empirically that it significantly outperforms a naïve approach based on regular expressions. We described several experimental results obtained by using the system for gazette creation for various resume related named entities (e.g., ORG, DEGREE, EDUCATIONAL_INSTITUTE, DESIGNATION) and the associated NER on a large set of real-life resumes. The proposed gazette creation algorithm exhibited significant precision and created large and quite complete gazettes for various named entities of interest in resume processing. Our gazette-based NER algorithm also works quite efficiently and accurately for NER using large gazettes over large resume repositories.

For further work, we are experimenting with using the algorithms in this paper to create gazettes for many other types of named entities found in resumes, such as technology skills, domains, awards, certifications etc. We are also looking at the problem of distinguishing between an instance that appears in multiple gazettes; e.g., Bank of Scotland may be mentioned in a resume either as a CLIENT or as EMPLOYER. We are considering how we can enrich the structure of a gazette to include more "knowledge" derived from the various occurrences of each instance across many resumes. We are also applying the system to create gazettes for many other types of named entities found in other kinds of documents such as financial reports. Finally, we are looking at enhancing the basic gazette creation algorithm to create gazettes for named entities which are organized in terms of a hierarchy; PERSON may be organized in terms of SPORTPERSON, SCIENTIST, ENTERTAINER, AUTHOR etc. which in turn may be further organized as ASTRONOMER, CHEMIST etc. In such cases, we need to not only identify an instance, but also the correct "level" of that instance.

ACKNOWLEDGMENTS

We are grateful to Prof. Harrick Vin, Vice President, Tata Consultancy Services Limited, for resources and discussions.

REFERENCES

[1] COLLINS, M. AND SINGER, Y. 1999. Unsupervised models for named entity classification. *Proc. EMNLP*.

[2] ETZIONI, O., CAFARELLA, M., DOWNEY, D., POPESCU, A.-M., SHAKED, T., SODERLAND, S., WELD, D.S. AND YATES, A. 2005. Unsupervised named-entity extraction from the Web: An experimental study. *Artificial Intelligence*, 165, pp. 91–134.

[3] NADEAU, D., TURNEY, P. AND MATWIN, S. 2006. Unsupervised named-entity recognition: generating gazetteers and resolving ambiguity. *Proc. 19th Canadian Conf. Artificial Intelligence*.

[4] PALSHIKAR, G.K., 2011. Techniques for named entity recognition: a survey. TRDDC Technical Report.

[5] THELEN, M. AND RILOFF E. 2002. A bootstrapping method for learning semantic lexicons using extraction pattern contexts. *Conference on Empirical Methods in Natural Language Processing (EMNLP 2002).*