RC22978 (W0311-056) November 7, 2003
Computer Science

IBM Research Report

Recognition of New Words Based on Entropy and Morphological Rules

Youngja Park
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Recognition of New Words based on Entropy and Morphological Rules

Youngja Park
young_park@us.ibm.com
IBM T.J. Watson Research Center,
19 Skyline Dr., Hawthorne,
NY 10562, USA.

October 26, 2003

Abstract

No lexicon could be expected to contain every possible word of a language, given the dynamic nature of languages and the creativity of human beings. Words unknown to the lexicon cause a lot of problems to natural language processing (NLP) systems which depend on lexical information such as part-of-speech tagging and terminology identification. Recent advances in technology speed up the creation of new, especially domain-specific, words; thus, timely and proper recognition of new words is very important for building reliable NLP systems.

This paper proposes methods for identifying probable real words among out-of-vocabulary (OOV) words in text and for generating possible parts-of-speech (POS) of the discovered new words. The identification of new words is performed based on the morphological rules of the language for derived new words and based on entropy of character trigrams for newly coined words. The POS guessing is done on the basis of lexical formation rules and word endings respectively. The proposed methods show promising results in both precision and recall.

1 Introduction

It is almost impossible to list in a dictionary all the words being used; thus, inevitably, there always exist OOV words[1] in documents. Due to recent advances in technology, the number of OOV words grows at faster rate. Words unknown to the lexicon cause a lot of problems to NLP systems which depend on lexical information such as POS taggers and parsers. In addition, the wrong interpretations of a tagger or a parser propagate to later applications deteriorating the performances. Therefore, proper and timely handling of OOV words is a crucial step for successful NLP applications.

[1] Also called as unknown words.

Much research aimed to address *unknown word problem* especially in the areas of POS tagging [2, 4, 24] and speech recognition [6, 7]. These systems guess the most probable POS (in POS tagging systems) or the closest substitute (in speech recognition systems) of an unknown word based on the context surrounding the word. For example, Hidden Markov Model (HMM)-based POS taggers decide the POS tag of an unknown word on the basis of POS distribution and the probability of POS transition in the training corpus.

However, those approaches presume all the OOV words are real words but only unknown to their lexicons or training data. This assumption causes problems in NLP applications such as terminology identification and machine translation. For instance, most taggers depend the POS assignment decision on the statistical information obtained from the given training data, and tend to assign noun tags to many unseen words. Due to POS taggers' inclination toward nouns, terminology identification systems are vulnerable to many arbitrary phrases.

In this work, we present unsupervised lexical acquisition methods for recognizing probable real words of a language. The goal of the algorithms is identifying possible real words among OOV words and suggesting their parts-of-speech. The system is especially useful for processing domain-specific technical documents since these documents tend to contain many new words. It improves the accuracies of POS taggers and terminology identification systems. This system is also a useful tool for lexicographers to automatically augment existing dictionaries or to build domain-specific dictionaries.

1.1 Types of OOV words

We classify OOV words into the following categories based on the analyses of a large collection of documents:

- derived words
- newly coined words
- proper nouns and abbreviations
- non-word strings

Derived words are morphological variations of words already known to the lexicon, mostly by means of affixation, i.e., adding prefixes to the beginning of words or suffixes to the end, and by means of compounding, i.e., two or more words are put together as one word [18]. Some examples[2] of derived words in English are *aerobiological*, *landmines* and *warzone*.

Newly coined words mean words that can not be produced by the derivation (or word formation) rules from existing words. As new technologies develop faster than before, more and more new technical jargon is being introduced. Examples of newly coined words are *anisotropy*, *eyellipse* and *ingress*.

[2]Examples of all the OOV words in this paper are based on the dictionary we use for the experiments. Some of these examples may exist in other dictionaries.

Proper nouns (mostly person and place names) and abbreviations are also common OOV words. We consider upper case and non-initial mixed case words as proper nouns or abbreviations.

Non-word strings include all other alphabetic strings and non-alphabetic strings (strings with characters such as numeric characters and other special characters). Email addresses, URLs, and some product names belong to this category.

1.2 Overview of the work

In this work, we focus only on the identification of derived words and newly coined words in text. For proper names and abbreviations, the author refers readers to previous work [19, 22] and [14, 16] respectively.

The overall process for identifying these new words and for producing lexical information are as follows.

First, we look up all the words appearing in a document in a general-purpose English dictionary [8], and regard words not found in the dictionary as OOV words.

Second, a filtering process is performed. If an OOV word is considered as an abbreviation, a proper name or a non-word string, the word is discarded. A named entity recognition [19] method is used for recognizing proper nouns. We developed a system for matching abbreviations and their definitions in text [16] and use the system in this work for filtering abbreviations. In addition, we also use capitalization information if the document is mixed-case text. If a word is in mixed case or upper case, we consider the word as a proper noun. Strings with one or more non-alphabetic characters are regarded as non-word strings and also excluded from further consideration.

Third, we test if an OOV word is comprised of an existing word and one or more morphological units such as prefixes and suffixes. If this process succeeds, possible parts-of-speech of the word are generated based on the morphological rules applied to produce the word.

Finally, if the morphological rule-based approach fails, we judge if the word might be a newly coined word on the basis of entropy of the probability of its character trigrams and guess its parts-of-speech based on its ending characters.

The rest of the paper is organized as follows. We present the morphological rule-based approach in section 2 and introduce an unsupervised method for finding affixes in section 3 respectively. The entropy-based approach is described in detail in section 4. In section 5, we show experimental results and evaluate the performance of the proposed method. Previous related work is described in Section 6. Finally, we discuss possible future improvements in section 7.

2 Morphological Rule Approach

This approach performs the recognition and POS guessing processes for OOV words given that the sub-components of the words are already known. We take into account two types of derived words—affixed words and compound words. The system takes traditional approaches for discovering derived words; that is, splitting an OOV word into its subcomponents and judging if all the subcomponents are known and the word formation is correct.

2.1 Affixed Words

Words with one or more prefixes and/or one or more suffixes are being used in text very often. An affix is any element in the morphological structure of a word other than a stem [21]. Affixes are divided into prefixes which come before the form to which they are joined and suffixes which come after.

When an OOV word is encountered, this algorithm first parses the word into a possible stem and one or more affixes. The affix(es) is looked up in our affix lexicons and the formation rule for the affix is retrieved. If the algorithm succeeds the parsing, the word is considered as a new derived word, and the possible POS tags are assigned according to the morphological rules described in the affix lexicons. We use pre-compiled lexicons of prefixes and suffixes for parsing words and for guessing part-of-speech tags of the words. The affix lexicons were built by processing a collection of documents (section 3 describes the affix finding algorithm). Currently, the prefix lexicon contains 244 prefixes and the suffix lexicon has 173 suffixes.

The process for prefixed words is as follows. First, the system checks if any of the prefixes in the prefix list appears at the beginning of an OOV word. If a word contains a prefix, then the system chops the prefix off the word and looks up the remaining part (the stem) in the dictionary. We set the minimum length of a stem to two characters. If the dictionary contains the stem, the OOV word is regard as a real word, and the word inherits the lexical information of the stem. For example, *anti*asthmatic and *auto*injector are discovered by the prefix process and regarded as an adjective and a noun respectively.

The process for suffixes is more complicated. Suffixes provide information about the syntactic categories to which words belong; for instance, if an English word ends with "ful", it is likely an adjective, and if a word ends with "ments", it implies the word is a plural noun.

Suffixation rules are recorded in the suffix lexicon. An entry in the lexicon includes all possible pairs of pre-conditions and the outcomes for a stem to have a specific suffix. The structure of suffix rules is as follows.

$$[\textit{suffix}, \{\ \textit{precondition-POS} \rightarrow \textit{result-POS}\ \}^*]$$

A suffix can have more than one pairs of precondition-POS and result-POS. For instance, the rule for suffix *able* is [*able*, {VB $\rightarrow$ JJ}, {NN $\rightarrow$ JJ}]. This rule means that a verb or a noun may have suffix *able* and the resulting part-of-speech is an adjective for both cases.

If a word contains a suffix, the system removes the suffix and recovers the root word. In English, when a suffix is added to a word, it may change the spelling of the stem. For instance, words ending with a silent *e* usually drop the *e* before a suffix beginning with a vowel. An example of this case is *browsable*. The final *e* of *browse* was dropped as a result of adding *able*. Thus, after separating a suffix from the stem, we recover the original form of the stem by using linguistic information. If the recovered stem is found in the dictionary and it has one of the pre-condition POS, then the word is regarded as a real word and it has the result POS in the rule. Some examples of this case are *migranious*, *oxidizability* and *ventilatory*. Some words, for example, *remanufacturability*, may have both a prefix(es) and a suffix(es). In this case, the word goes through both prefix and suffix processes.

2.2 Compound words

A compound word is made up of two or more words that together express a single idea [18]. We define a compound word as a combination of two content words. The types of compound words and the possible combinations of two POS tags are as follows.

- a compound noun formed with noun+noun (*audiotape, filename, eyedrops*)
- a compound noun formed with adjective+noun (*hardliner, lightweight*),
- a compound noun formed with noun+gerund (*airbreathing, housekeeping, stockholding*)
- a compound adjective formed with adjective+noun+{-d/-ed} (*doubleblinded, lightheaded, singleminded*),

The steps for processing compound words are as follows:

1. recognize the boundary of two components
2. check if the compounding of the two components is valid (i.e., one of the four cases above)
3. assign the POS of the head component to the compound word

The system splits a word into two parts in an exhaustive way. We set the minimum length of a component of a compound to 2 letters. Thus, for a word with n ($n \geq 4$) letters, we examine $n-4+1$ possible splits. Two components of a split are looked up in the dictionary. If both components of the split exist in the dictionary and their parts-of-speech satisfy one of the predetermined valid combinations, the candidate compound is considered as a real word and the POS of the head component is assigned to the new word.

3 Automatic Affix Discovery Algorithm

Note that the algorithm described in section 2 requires well-compiled prefix and suffix dictionaries. Like content words, new affixes are also being introduced

and many, especially domain-specific, affixes are absent from dictionaries. For instance, 150 of 219 prefixes found in a biomedical document collection are domain-specific prefixes as shown in Figure 1.

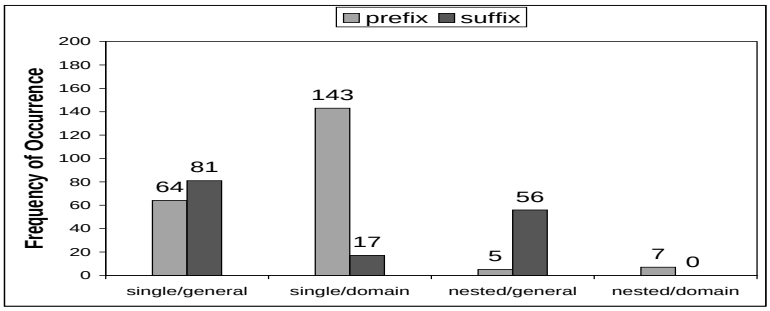

Figure 1: Types of affixes in MEDLINE. Four categories of affixes: single and general affixes, single and domain-specific affixes, nested and general affixes, and nested and domain-specific affixes. In this Figure,'single' means one prefix (and/or suffix) is attached to a word (for example, *co*-sponsor-*ship*) and 'nested' affix means more than one prefixes (or suffixes) used together for a word (for example, *radioimmuno*-assay and sleep-*lessness*).

We propose an unsupervised method for finding affixes from a collection of documents. This tool is useful especially for technical languages, which usually don't have morphology dictionaries. We identify affixes by using an iterative optimization process of the discovery of prefixes and suffixes. The discovery of affixes are based on two facts—sharedness and stemhoodness. Sharedness denotes affixes are shared by many words, and stemhoodness means the rest of the word after splitting off affixes should be a good stem.

3.1 Sharedness

A primary characteristic of prefixes and suffixes is that they are shared by many different words. In this work, we define word-initial (word-final) substrings that appear in at least two different words as *potential prefixes (suffixes)*. To easily find the degree of sharedness of strings, we represent words in documents by using Patricia (Practical Algorithm to Retrieval Information Coded in Alphanumeric) trees [15]. A Patricia tree is a compact representation of a trie and can be constructed by compressing all unary paths in a trie.

Every word in the input collection is inserted into a Patricia tree, and we call it a prefix Patricia tree. Then, the word is reversed and inserted in a suffix Patricia tree. Since a suffix of a string is a prefix of the reversed form of the string, the problem of finding suffixes is reduced to the problem of finding

prefixes. Thus we re-use the data structure and the algorithm designed for the prefix discovery for the suffix discovery as well.

Figure 2 shows the prefix Patricia tree and the suffix Patricia tree for eight words: *anti-anxiety*, *anti-cancer*, *antioxidative*, *cardioacceleratory*, *cardioactive*, *cardiography*, *neurocognitive*, and *neurologist*.

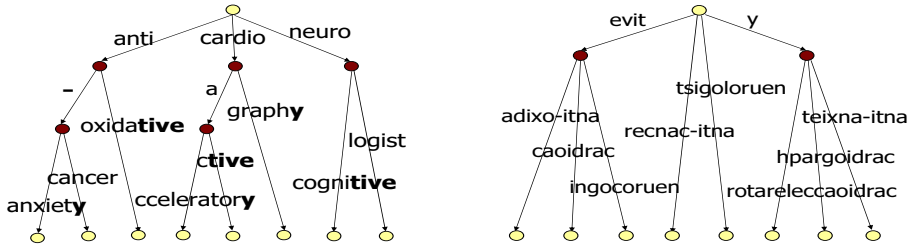

Figure 2: Prefix and suffix Patricia trees for words: anti-anxiety, anti-cancer, antioxidative, cardioacceleratory, cardioactive, cardiography, neurocognitive, neurologist.

According to the definition of Patricia tree, all non-terminal nodes (dark nodes in Figure 2) in a Patricia tree have at least two branches. In other words, the key strings for non-terminal nodes are shared by at least two different words; that is, these strings are potential affixes. In the example of Figure 2, *anti*, *anti-*, *cardio*, *cardioa*, and *neuro* are regarded as potential prefixes, and *tive*, *y* are regarded as potential suffixes. Note that there may be multiple non-terminal nodes on a path from the root node to a terminal node. Thus a word can have multiple potential prefixes and suffixes. For instance, *cardioacceleratory* has two potential prefixes–*cardio* and *cardioa*.

3.2 Stemhoodness

Not all potential affixes are valid affixes; some potential affixes are not affixes at all even though they appear as a prefix or a suffix of many other words (e.g., *ag* as in *again* and *agent*, or *ack* as in *attack* and *feedback*). A potential affix is considered as a valid affix if the rest part of the word is a valid word (a simple word or an affixed word). In other words, the goodness of a potential affix is determined based on the stemhoodness of the potential stems.

In this work, a potential stem is regarded as a good stem if it occurs as a free-standing word (not as part of a longer word) in the given document collection or a valid affixed word. We take into account two factors to compute the goodness of an affix: the number of good stems and the ratio of good stems. Let S be the set of all the potential stems of an affix a and S' be the set of good stems of a. Then, the number of good stems, $C_a(S')$, and the ratio of good stems, $R_a(S')$ are defined as in equation 1.

$$C_a(S') = \mid S' \mid, \quad R_a(S') = \frac{C_a(S')}{\mid S \mid} \tag{1}$$

Let CP be the set of candidate prefixes and CS be the set of candidate suffixes.
1. W = {w | w is a word in the collection}, CP = {}, CS = {}
2. For each potential affix, find the set of good stems (S').
   A potential stem r is a good stem if it satisfies one of the two conditions.
   (1) $r \in$ W, or
   (2) r = [p'] w[s'], where $p' \in$ CP, $w \in$ W and $s' \in$ CS
   [p'] and [s'] denote that p' and s' are optional.
3. Compute the number and the ratio of good stems as in equation 1
   and extend CP and CS with new candidate prefixes and suffixes.
4. Repeat from step 2 until no new prefixes or suffixes are found

Table 1: Algorithm for finding prefixes and suffixes

If $C_a(S')$ is greater than a give threshold T_1 and $R_a(S')$ is greater than a threshold T_2, then the affix a is considered as a candidate affix. T_1 and T_2 are adjusted based on our experiments and are set to 2 and 0.25 respectively.

The algorithm for finding affixes is shown in Figure 1.

4 Entropy-based Approach

Humans can successfully guess whether a word never-seen before is a possible real word or not, even though the word is not comprised of already known components. We assume that people conclude a word is a possible real word if the sequence of characters in the word looks probable and it is natural to pronounce. In this work, we base the recognition of non-derivational new words on this assumption. More formally, a word is considered as a cohesive group of letters with strong internal statistical influences. The problem of judging if a sequence of letters is a real word is defined as the problem of measuring the degree of our surprise on seeing each letter at its position.

Entropy is the average uncertainty of a random variable [20]. In this work, we use entropy as our measurement for deciding possible real words. An OOV word is regarded as a possible real word if the entropy of the word is relatively low (i.e., not a big surprise). If a model captures more of the structure of a language, then the entropy of the model should be lower.

We discuss our entropy model in more detail in section 4.1, and describe about the training method and the training data in section 4.2. This system also suggests the part-of-speech of the newly coined word based on the word-ending letters. The process for guessing POS tags is explained in section 4.3.

4.1 Entropy model

The entropy model computes the average surprise of letters in a word appearing in their locations when the neighbors have seen. The smaller the surprise is, the more likely the word is a real word.

Let us first define the probability function used in our model. The probability function measures the chance of seeing a letter c with a given history h; that is, $p(c|h)$. The history can be from one previous letter to all the letters before c in a given document. Determining an appropriate length of the history depends on the applications and the size of training data, and the following tradeoff needs to be considered. The longer the history is, the more information it provides; thus, the prediction becomes more accurate. On the other hand, since the training data is limited, longer history encounters more serious data-sparseness problem deteriorating the system performance. In this model, we use a trigram model; that is, the probability of a letter depends only on its previous two letters.

We apply maximum likelihood model (MLE) to estimate the probability of an n-gram model as shown in equation 2.

$$P_{MLE}(c_n \mid c_1 \ldots c_{n-1}) \;=\; \frac{f(c_1 \ldots c_n)}{f(c_1 \ldots c_{n-1})} \tag{2}$$

where $f(c_1 \ldots c_n)$ denotes the frequency of an n-gram in the training data. MLE makes the probability of observed events as high as it can and does not waste any probability mass on un-seen events. However, MLE can not produce probabilities for the unseen events (data-sparseness problem).

Many approximation methods have been introduced to reduce the effect of the data sparseness in the training data [11]. We use deleted interpolation method which combines lower order, more reliable, n-gram predictions to estimate a higher order n-gram prediction [3, 10]. This is generally done through a linear interpolation process wherein predictions from each n-gram order are weighted by a different factor.

The final probability function for a trigram, $c_{i-2}\ c_{i-1}\ c_i$, is defined as Equation 3.

$$P(c_i \mid c_{i-2}, c_{i-1}) = \lambda_1 P_{MLE}(c_i \mid c_{i-2}, c_{i-1}) + \lambda_2 P_{MLE}(c_i \mid c_{i-1}) + \lambda_3 P_{MLE}(c_i) \tag{3}$$

where $0 \leq \lambda_i \leq 1$ and $\sum \lambda_i = 1$. λ_i is a linear weight for specifying the contribution of each n-gram.

Summing over the surprise of the predictor at each letter gives an expression for our total surprise (i.e., entropy). The pointwise entropy of a word w, $H(w)$, is defined as in Equation 4.

$$H(w) = -\sum_{i=1}^{n} log_2\ P(c_i \mid c_{i-2},\ c_{i-1}) \tag{4}$$

We normalize the entropy by the length of the word so that the surprise is not dependent on the size of the word. Finally, we decide a word as a real word if the normalized entropy is lower than a given threshold θ.

$$\frac{H(w)}{\mid w \mid} \leq \theta \tag{5}$$

The threshold value is set to 4.2, which was computed by the mean entropy plus the standard deviation of all words in the training corpus.

4.2 Model Training and Testing

The entropy model can be trained from any set of words, which can be considered as the representatives of the target language. Applications can choose their own training data according to the domains and the purposes. In this work, we use a general-purpose English dictionary [8] as training data. The dictionary contains 46,247 base forms. We removed short words with the length less than 4, and generated all the inflectional forms of the words in the dictionary, The final training set comprises of 79,644 word forms. The training data contain some foreign words, person names and abbreviations, but these words don't give substantial influence on trigram statistics.

To compute $P_{MLE}(c_i \mid c_{i-1}c_{i-2})$ for $i = 1 \ldots n$ (equation 2), we need to introduce two new letters—a beginning dummy letter and an ending dummy letter. Thus English becomes a 28 letter alphabet language. Through the paper, we denote the beginning dummy letter and the ending letter as @ and $ respectively. For each word in the training data, two beginning dummy letters and two ending dummy letters are added; that is, a word w with n characters, $c_1\, c_2 \cdots c_n$, is extended to $w^{'} = c_{-1}\, c_0\, c_1 \cdots c_n\, c_{n+1}\, c_{n+2} = @@\, c_1\, c_2\, \cdots c_n\, \$\$$.

The training module performs the following steps:

1. segments all the words in the training data into unigrams, bigrams, and trigrams.
2. counts the frequencies of all unigrams, bigrams and trigrams found in the training data.
3. computes the probabilities of unigrams, bigrams and trigrams by using equation 2. Note that $P_{MLE}(c_1 \mid @@) = P_{MLE}(c_1 \mid @)$ and $P_{MLE}(\$ \mid c_n\, \$) = P_{MLE}(\$ \mid c_n)$.
4. saves the probabilities in a lookup table.

Our training data generated 28 unigrams, 566 bigrams and 5894 trigrams. Note that we have data sparseness problem, even though it is less serious than other NLP applications. Only 30 % of possible trigrams (5894/19683) are present in the training data whereas about 78 % of possible bigrams and 100 % of unigrams are found in the training data.

In a testing phase, the testing module also augments a given OOV word with the beginning and the ending dummy letters. It then produces trigrams in the word and retrieve the probabilities from the n-gram probability lookup table. The entropy of the word is computed by equation 4, and the word is regarded as a real word if the entropy is higher than the given threshold.

4.3 POS guessing for new words

In addition to identifying non-derivational new words, this system assigns possible parts-of-speech to newly coined words. We adopt the ending guessing method described in [13] for this purpose. The ending guessing rules are collected from the training dictionary. For all the words in the training data, we

generate all possible endings from length 1 up to length 5, together with the parts-of-speech of the words. We set the minimum length of the remaining part to 3.

Table 1 shows how ending guessing rules are generated from our training data. Throughout this paper, POS tags are represented by Penn Treebank Tag code [12].

word	ailments	mounting		abandons	primary	
Ending Rules	ments NNS	nting NN	nting VBG	ndons VBZ	mary NN	mary JJ
	ents NNS	ting NN	ting VBG	dons VBZ	ary NN	ary JJ
	nts NNS	ing NN	ing VBG	ons VBZ	ry NN	ry JJ
	ts NNS	ng NN	ng VBG	ns VBZ	y NN	y JJ
	s NNS	g NN	g VBG	s VBZ		

Table 2: Examples of Ending Guessing Rules

All the ending rules and their frequencies are collected from the training data. We discarded infrequent rules (frequency = 1) from the rule set resulting in 12,387 rules. Table 3 shows the most frequent 50 rules. The numbers in parentheses denote the frequencies of the rules.

s	NNS	(19301)	g	VBG	(7116)	ng	VBG	(7111)
ing	VBG	(7075)	s	VBZ	(7006)	d	VBD	(6879)
ed	VBD	(6768)	es	NNS	(5119)	y	RB	(4701)
ly	RB	(4668)	rs	NNS	(4336)	r	NN	(4315)
e	NN	(3562)	ers	NNS	(3462)	es	VBZ	(3436)
er	NN	(3423)	s	NN	(3423)	ss	NN	(3179)
ess	NN	(3136)	ness	NN	(3068)	n	NN	(2843)
e	JJ	(2773)	e	VB	(2763)	t	NN	(2498)
y	NN	(2490)	ts	NNS	(2422)	on	NN	(2026)
ns	NNS	(1988)	t	JJ	(1858)	ion	NN	(1803)
d	JJ	(1725)	tion	NN	(1569)	ting	VBG	(154)
ted	VBD	(1482)	l	JJ	(1451)	ed	JJ	(1408)
ons	NNS	(1391)	ies	NNS	(1330)	le	JJ	(1323)
y	JJ	(1297)	ble	JJ	(1238)	lly	RB	(1237)
al	JJ	(1212)	ation	NN	(1187)	ions	NNS	(1185)
ty	NN	(1184)	r	JJ	(1152)	ity	NN	(1084)
ally	RB	(1075)	able	JJ	(1056)			

Table 3: 50 most frequent ending-guessing rules in the dictionary

By using the rule set, the system produces all possible parts-of-speech of a word on the basis of the longest matching pattern. We look up the ending letters of the word in the rule set from the longest ending (5 letters if the word's length is larger than 7, otherwise the word's length minus 3) to the ending of length 1 (the final letter). If an ending exists in the rule set, the matching process stops, and the system produces all the parts-of-speech of the ending in the order of the rule frequencies. For instance, *cortical* is guessed as an adjective

and a noun, but adjective reading is preferred because *tical* appears 105 times as an adjective and 4 times as a noun in the training data.

5 Performance Evaluation

The evaluations of the presented algorithm were conducted in two ways; (1) validation of the discovered new words, and (2) the improvement of a POS tagger's precision with the suggested parts-of-speech of the new words. The first method examines how many real words are judged to be new words by the proposed system. The second method checks how much a POS tagger benefits from the POS suggestions by the system.

Two domain-specific document collections were used for the evaluations — a WMD (Weapons of Mass Destruction) document collection and a MEDLINE abstract collection. The MEDLINE collection is about 6 mega-bytes in size and contains 4,000 abstracts. We selected medical documents for the experiment because many of the medical terms are OOV words but they are relatively easily verifiable because many medical dictionaries exist . The WMD collection is about 2 mega-bytes in seize and consists of about 8070 WMD glossary definitions Table 4 shows the detailed types and the frequencies of the OOV words found in the test collections.

OOV Type	*MEDLINE*	*WMD*
Affixation	1463	395
Compound	74	160
coined word	3336	506
Misc.	679	667
Total	5552	1728

Table 4: Number of OOV words in MEDLINE and WMD collections with respect to OOV types

5.1 Validation of the OOV words

We validate affixed words, compound words and newly coined words in Table 4 (4873 words for the MEDLINE collection and 1061 words for the WMD collection) if they are in fact real words. To verify the system's judgment, we use two on-line dictionaries— a general-purpose English dictionary[3] and an on-line version of the Merriam-Webster medical dictionary[4]. The reason for selecting a medical dictionary is most OOV words in our test data are medical vocabulary. First, we look up all the suggested new words in the on-line English dictionary. Then, words which do not exist in the dictionary are looked up in the medical dictionary.

[3] http://www.dictionary.com

[4] http://www.intelihealth.com/IH/ihtIH/WSIHW000/9276/9276.html

We developed Perl [23] script programs to automate the dictionary look-up processes. The programs perform dictionary look-up with a URL and a word without any human intervention. They access the web-page of the given URL, perform search with the given word, and return the web-page of the search result. Then, they parse the returned web-page and decide if a word was found or not.

Table 5 and Table 6 show the results of this evaluation for the MEDLINE collection and the WMD collection respectively. The first column (*Dictionary Lookup-Yes*) denotes the number of the words found in one of the dictionaries, and the second column (*Dictionary Lookup-No*) denotes the number of the words which don't exist in any of the dictionaries. The first row (*System Guess-Yes*) denotes the number of the words which our system considered as real words, and the second row (*System Guess-No*) is the number of the words which the system regarded as invalid words.

		Dictionary LookUp		
		Yes	No	total
System	Yes	3377	710	4087
Guess	No	536	250	786
	total	3913	960	4873

Table 5: Dictionary-Lookup Results for the MEDLINE collection

		Dictionary LookUp		
		Yes	No	total
System	Yes	699	220	919
Guess	No	72	70	142
	total	771	290	1061

Table 6: Dictionary-Lookup Results for the WMD collection

Table 7 shows the precision, recall and F-measure of the experimental results.

	MEDLINE	WMD
precision	82.63%	76.06%
recall	86.21%	90.66%
F-measure	84.38%	82.76%

Table 7: Dictionary-Lookup Results for the WMD collection

Note that many of the samples that the system decided real words but not found in the dictionary (*System Guess-Yes* and *Dictionary Lookup-No*) are actually valid words. This is because the dictionaries used for this experiment are also limited. Some examples of these words — mostly biology terminology and

drug/treatment names — are *aggregometry*, *cardiomyocyte*, *colforsin*, *nondihydropyridine*, *nylestriol*. Thus, we expect the actual recall and precision of the system would be higher than the figures in Table 7.

5.2 Improvements of a POS tagger

The second experiment checks how much the presented methods improve the accuracy of a POS tagger. We conducted this test on a subset of the WMD collection and a subset of our MEDLINE abstracts collection (50 abstracts). WMD data has 34,187 tokens and the MEDLINE data contains 14,719 tokens.

The experiment was conducted as follows. We used a HMM-based tagger in the Talent System [9] which has no special unknown word treatments. The tagger randomly guesses a POS tag for an unknown word.

We first ran the tagger alone (*run1*) and saved the results. The same tagger was run again with the presented system (*run2*). We changed the tagger to take the POS suggestions by this system, and to decide a POS tag among the suggestions. The two runs produced different POS tags for 265 tokens for the WMD data and 560 tokens for the MEDLINE data. Due to the time required for manual evaluation, only these different POS tags were manually examined. Table 8 shows the experimental results. The first row shows the number of tokens for which *run1* produced correct tags, and the second row shows the number of tokens for which *run2* assigned correct POS tags. The third row shows the number of tokens for which both runs fail to assign correct tags. Even the size of test is small, we can see the algorithm can help POS taggers to deal with OOV words correctly.

	WMD	MEDLINE
correct tags by *run1*	8	20
correct tags by *run2*	239	482
incorrect tags by both	18	58
total tag differences	265	560

Table 8: Comparison of tagger performances with (*run1*) and without the presented system.

6 Related Work

To our knowledge, there is no previous work which can automatically decide if an unknown word is a valid new word of a target language. However, there has been many efforts to address unknown words in POS tagging systems. In this section, we discuss some of previous efforts for unknown words and other character ngram-based applications.

POS taggers heavily rely on lexical information about words and the goal of processing unknown words is to guess the most plausible POS tags and other lexical information of unknown words in a given context. Unknown words are

defined as words not existing in the systems' lexicons or never seen in the training data.

Dermatas and Kokkinakis [4] estimated the probability that an unknown word has a particular POS tag from the probability distribution of words which occur only once in the previously seen texts. This simple probabilistic approach can be easily trainable and more accurate than the naive random selections. More advanced POS guessing methods use orthographic features or lexical cues in leading and trailing word segments to determine possible POS tags for unknown words. Weischedel and Palmucci [24] proposed a POS guessing method for unknown words by using the probability for an unknown word to be of a particular POS tag, given its capitalization feature and its ending. Brill [2] describes a transformation-based approach for tagging unknown words. The rules use orthographic cues and also information about affixes. One example of the transformation rules is "change the tag from NN to NNS if the word has suffix *-s*". Mikheev [13] presents a technique for fully automatic acquisition of rules which guesses possible POS tags for unknown words using their starting and ending segments. The learning was performed from a general-purpose dictionary and word frequencies collected from a corpus.

Character ngram-based statistical approaches have been used in word-level language processing such as spell correction, word segmentation, and language identification. Angell *et al.* [1] describe a nearest neighbor search for automatic correction of misspellings. It replaces a misspelt word with the word in a dictionary which best matches the misspelling. The degree of match is calculated using a similarity coefficient based on the number of trigrams that the two strings have in common. Juola *et al.* [17] presents a system which segments full words into their constituent morphemes. The method bases the segmentation of a word only on entropy of the probabilities of trigram sequences in the word. The assumption is that the entropy (unpredictability) of a string of letters in running linguistic text achieves some sort of a maximum at the boundaries between morphemes. Dunning [5] implements a language identification system using a character n-gram model and a Bayesian classifier. The method does not use any language-specific rules but frequency of character n-grams. The algorithm performs well with only a modest amount of training text known to be in the languages of interest.

7 Conclusions

We have described unsupervised approaches for lexical acquisition in the context of finding new words—newly derived words and domain-specific technical words—through text analysis of document collections. These algorithms are useful not only for NLP applications but also for lexicographers for augmenting dictionaries.

For morphologically derived words, we employ morphological rule-based methods such as affixation and compounding. The morphological rule-based approach processes words with high precision but this method is inherently lim-

ited by the size of the affix lexicons. In other words, if an exhaustive list of affixes are available, this method can find most affixed new words with high confidence.

In this work, we presented an unsupervised learning algorithm for finding new affixes from unannotated text. This method uses Patricia trees for discovering potential affixes and an iterative optimization process to decide good affixes. This is a useful tool to collect domain-specific affixes which many dictionaries overlook.

The morphological rules for the compound word process in this work are somewhat over-generalized. For example, all the combinations of two nouns may not be compound nouns. We anticipate the performance will be improved if we incorporate a corpus statistic-based compound word processing scheme into the existing method.

We also proposed a new technique to identify non-derivational new words based on entropy of the probabilities of trigram sequences. In this work, the probabilities of trigram sequences are trained on an existing English dictionary. However, some domains such as biomedicine and chemistry have many words originated from foreign languages. Domain-specific dictionaries or a tagged corpus (annotated if a word is correct or not) can be good training data. These resources may represent the characteristics of the domain vocabulary better than general-purpose language dictionaries and thus improve the performance of the entropy-model.

It is not difficult to apply the approaches presented in this paper to other languages because this system only uses basic morphological rules of a language and language-independent statistical information. In addition, the proposed n-gram model requires only a small amount of un-annotated training data; making the algorithm easily adaptable across domains and languages.

References

[1] Angell, R., G. Freund, and P. Willett: 2002, 'Automatic spelling correction using a trigram similarity measure'. *Information Processing and management* **19**(4), 255–261.

[2] Brill, E.: 1995, 'Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging'. *Computational Linguisitics* **21**(4), 543–565.

[3] Chen, S. F. and J. Goodman: 1996, 'An empirical study of smoothing techniques for language modeling'. In: *Proceedings of the 34th Association for Computational Linguistics*. pp. 228–235.

[4] Dermatas, E. and G. Kokkinakis: 1995, 'Automatic stochastic tagging of natural language texts'. *Computational Linguisitics* **21**(2), 137–164.

[5] Dunning, T.: 1994, 'Statistical identification of language'.

[6] Gallwitz, F., E. Noeth, and H. Niemann: 1996, 'A category based approach for recognition of out-of-vocabulary words'. In: *Proceedings of International Conference on Spoken Language Processing.* pp. 228–231.

[7] Hazen, T. J. and B. Issam: 2001, 'A comparison and combination of methods for oov word detection and word confidence scoring'. In: *Proceedings of ICASSP.*

[8] IBM: 2001, 'IBM Dictionary and Linguistic Tools'.

[9] IBM T. J. Research: 2001, 'The Talent (Text Analysis and Language Engineering) Project'. http://www.research.ibm.com/talent/.

[10] Jelinek, F.: 1990, 'Self-organized language modeling for speech recognition'. In: A. Waibel and K. Lee (eds.): *Readings in Speech Recognition.* pp. 450–506.

[11] Manning, C. D. and H. Schuetz: 1999, *Foundations of Statistical Natural Language Processing.* The MIT Press.

[12] Marcus, M. and M. B. Santorini: 1993, 'Building a large annotated corpus of English: the Penn Treebank'. *Computational Linguistics* **19**(2), 313–330.

[13] Mikheev, A.: 1997, 'Automatic rule induction for unknown word guessing'. *Computational Linguistics* **23**(3), 405–423.

[14] Mikheev, A.: 2002, 'Periods, Capitalized Words, etc'. *Computational Linguistics* **28**(3), 289–318.

[15] Morrison, D. R.: 1968, 'PATRICIA–Practical Algorithm to Retrieve Information Coded in Alphanumeric'. *Journal of the ACM* **15**(4), 514–534.

[16] Park, Y. and R. J. Byrd: 2001, 'Hybrid Text Mining for Matching Abbreviations and their Definitions'. In: *Proceedings of Empirical Methods in Natural Language Processing.* pp. 126–133.

[17] Patrick, J., C. H. Skene, and A. Boggs: 1994, 'Corpus-Based Morphological Segmentation by Entropy Changes'. In: *The third International Conferences on the Cognitive Science of Natural Language Processing.*

[18] Pickett, J.: 1996, *The American heritage book of English usage: A practical and authoritative guide to contemporary English.* Houghton Mifflin Company.

[19] Ravin, Y., N. Wacholder, and M. Choi: 1997, 'Disambiguation of proper names in text'. In: *17th Annual ACM-SIGIR Conference.*

[20] Shannon, C. E.: 1951, 'Prediction and entropy of printed English'.

[21] Sproat, R.: 1992, *Morphology and Computation.* MIT Press, Cambridge, MA.

[22] Uchimoto, K., Q. Ma, M. Murata, H. Ozaku, and H. Isahara: 2002, 'Named Entity Extraction Based on A Maximum Entropy Model and Transformation Rules'. In: *Proceedings of the 38th Annual Meeting of Association for Computational Linguistics*.

[23] Wall, L. and R. L. Schwartz: 1992. O'Reilly & Associates, Inc.

[24] Weischedel, R. and J. Palmucci: 1993, 'Coping with ambiguity and unknown words through probabilistic models'. *Computational Linguistics* **19**(2), 359–382.