Kiwi: A Multilingual Usage Consultation Tool based on Internet Searching

Kumiko TANAKA-Ishii and **Masato YAMAMOTO** and **Hiroshi NAKAGAWA**
Language Informatics Laboratory, Information Technology Center
The University of Tokyo
{kumiko, yamamoto, nakagawa}@r.dl.itc.u-tokyo.ac.jp

Abstract

We present a usage consultation tool based on Internet searching. When a user enters a string of words that he wants to find the usage for, the system sends a query to the search engine to obtain a corpus about the string. The corpus is statistically analyzed and results are displayed. As the system uses neither language-dependent analysis nor initial data, queries can be made in any language, even languages for which there are no well-established analytical methods. Also, since the corpus is dynamically obtained from the search engines, the usages provided to the user are always up to date. Kiwi can fill in the missing parts of the collocations frequently used by native speakers.

1 Introduction

When a learner gets stuck on a question of usage, one possible solution is to use a search engine. For example, suppose that a Japanese student wants to find how to say "wireless network" in French. In Japanese, the most common way of saying "wireless network" is "musen (wireless) LAN". A direct translation cannot be found in any Japanese-French dictionaries, so typically the student will first try to translate part of the target using a conventional dictionary; if the student looks up "musen", the dictionary definition will be "sans fil". ("LAN" cannot be found in dictionaries, either). This clue, "sans fil", can then be input to the search engine, and usages such as "l'access sans fil" or "le reseau sans fil" will be retrieved within the first two or three pages of search-engine results.

The user can find out more if he scans, say, 50 pages of results. For example, he would find that the most common usage of wireless network is in the expression "l'internet sans fil" (followed by "les reseaux sans fil"). To make this task of scanning and summing up more practical, we need a tool that can read the entire 50 pages on behalf of the user since the process would otherwise be too time-consuming. For this purpose, we have developed Kiwi, a tool that dynamically obtains data from the worldwide web (WWW), scans tens of pages of search-engine results, statistically analyzes the pages, and reports the results to the user.

Internet search engines are increasingly used by language students to learn about usage, and the idea of creating a tool to facilitate this is not new. One example of an existing tool is Webcorp (Webcorp, 1999). Using the WWW as its corpus, Webcorp traverses links to obtain an extensive collection of relevant pages, and then it totals the search engine results. However, this tool is only available in English and is not readily applicable to languages without word segmentation. Another recent examples are found in (GoogleFight, 2000)(Peters, 2002), where the user enters two different phrases and the tool suggests which is most commonly used according to data obtained from the search engine. Although these pages allow multilingual consultation, the range of possible queries is limited because the two phrases can be compared only in terms of occurrence frequency.

Our tool, Kiwi, is designed to be more general in its applicability than these existing tools in that it is language independent and allows greater query flexibility. It does this through candidate extraction and candidate ranking.

Another concern that we address in this paper is to what extent the search-engine results can be used as an example-based language source. In this sense, our study is similar to that of (Keller et al., 2002). They argue that high-quality n-grams can be derived from the ostensibly-noisy WWW, although their approach is word based thus presumes the use of tagger for non-segmented languages. As we believe that the idea is more important to be utilized independently of language, we discuss the performance when the idea is applied to the string-based processing. Such results can serve as the fundamental study useful for more difficult tasks based on the use of search engines: for example, Question and Answering (Brill et al., 2001) (Dumais et al., 2002) or dynamic term translation(Nagata et al., 2001). In the next section, we explain Kiwi in more detail.

2 KIWI

Kiwi is a simple tool written in the Java language. It is designed for use with an Internet connection. The Kiwi processing is as follows:

1. Receive an input query from a user:
2. Obtain a fixed number of search engine results that match the query.
3. Extract all fixed length strings that might include candidates.
4. Obtain candidates from these strings.
5. Rank candidates.
6. Display results.

Compared with previous works (Webcorp, 1999) (GoogleFight, 2000), Kiwi allows more flexible query entries.

Wild Card '*' can be used to replace a missing word or string. For example, "human *", "* sans fil", can be entered into Kiwi.

Comparison allows us to compare two or more phrases and prioritize them depending on which is most relevant. This can be expressed as (A/B/C...).

Another feature of Kiwi is language independence. We adopted this in order to widen opportunities of usage consultation, even in languages not widely used internationally.

Let's look at some actual examples of how Kiwi can be used(Table 1). These examples are related to seven languages. Some involve the translation of "wireless network", and the others show VIP names, film titles, food names, and so on. We see that the results directly reflect our times: we watch Harry Potter and the Sorcerer's Stone and Star Wars, eat Kimchi and Peking duck, visit Neuschwanstein schloss. Such

Table 1: Kiwi Live Examples

Input	Kiwi Result Examples	
English		
wireless*	communications	network
Harry Potter*	and the Sorcerer's Stone	and the Chamber of Secrets
*animation	flash	cartoon
Bovine*	Spongiform Encephalopathy (BSE, mad cow disease)	
Star*	Wars	Trek
French		
*sans fil (wireless)	réseaux (network)	téléphone (telephone)
*du monde (of world)	coupe (cup)	tour (tour)
*Zidane (Soccer player)	Zinedine (first name)	Zinédine (first name)
Japanese		
無線* (wireless)	LAN (LAN)	LAN カード (LAN card)
東京* (Tokyo)	スタジアム (Stadium)	ディズニーランド (Disney Land)
*純一郎 (Jun'ichiro)	小泉 (Koizumi)	内閣総理大臣小泉 (Prime Minister)
Chinese		
无线 * (wireless)	网络* (network)	电管理 (radio management)
*拳 (Martialart)	太极 (Tai Chi)	太極 (Tai Chi)
*烤鴨 (duck)	北京 (Beijing)	全聚德 (restaurant name)
Korean		
*김치 (kimchi)	포기 (Pogi kimchi)	배추 (Chinese cabbage)
German		
schloß* (castle)	Neuschwanstein (castle name)	Neuhaus (castle name)
Spanish		
Real* (Soccer team)	Madrid (team name)	Sociedad (team name)

Table 2: Top 10 Altavista Results for 'wireless'

1st	Pre-release wireless Java software
2nd	Beta-edition, wireless Java software
3rd	All digital wireless
4th	AT&T wireless
5th	AT&T wireless
6th	on the wireless Internet
7th	5-day wireless boot-camp
8th	Wireless Lans
9th	Wireless phone calling plans
10th	Wireless News

live results can be observed only with a simple tool such as Kiwi that sums up the overall trend of the search engine data.

What we want to emphasize here is that such an overall view cannot be obtained through the direct use of a search engine by the user. Table 2 shows the Altavista search result for "wireless". (Kiwi currently uses Altavista as its mother search engine because of its indexing strategy). "wireless Internet" had the sixth highest score, and "Wireless Lans" had the eighth highest. This result is badly affected by noise. By comparison, Kiwi gave far clearer results: *communications*, *network*, and *Internet*.

As explained in §1, the two most important functions of Kiwi are the candidate extraction and the candidate ranking. We next explain the design of these functions in Kiwi.

3 Candidate Extraction and Ranking

Intuitively, we would expect a candidate to have the following features:

A It occurs frequently.

B It has moderate length.

C There is a variety of succeeding characters.

A concerns the relevancy of the candidate (as will be discussed in the next section). On the other hand, **C** essentially expresses information about the word boundaries. **B** can be used for either objective.

Combining **C** and **B**, we decided to extract the candidates in the following way. With X as a string, let X_i be the head i characters of X, and C_i be the number of *kinds* of characters at the $(i+1)$th position of all strings of the prefix X_i. We extract X_i as a candidate when:

$$C_i > C_{i-1} .$$

Such idea of using the branching degree has been proposed in a various way (resumed in (Nakagawa and Mori, 2002)) and shown to be successful. What is new of our idea is to apply their idea to string level, that is to say, how many kinds of characters suceed the focused chracter.

After candidates are obtained, Kiwi ranks them. Here, the most important information is frequency (**A**). The length is also a concern, though, because short strings obviously occur more frequently than longer ones (condition **B**). Therefore, a bias towards longer candidates should be included in the evaluation function. The question is how best to incorporate these two features into an evaluation function formula.

We decided to do this empirically. With X as the candidate, $|X|$ as its length, and $freq(X)$ as its frequency, we used the following function as our evaluation function:

$$F(X) = freq(X)log(|X| + 1).$$

We chose to use this function because it outperformed all other functions we considered.

4 Evaluation

4.1 Data and Process

We evaluated Kiwi when using the Altavista search engine. English collocations were chosen as our query test set because hundreds of them were automatically available. We here show only of the evaluation on cases where the wild card was placed at the head or the end of the query.

Our evaluation was done in five stages:

1. Obtain a set of collocations with each being more than three words in length.
2. For each collocation, replace the head or the tail part with the wild card. These are called *queries*, and the replaced words are called *required answers* in the following.
3. Scan all the queries and remove ambiguous queries (such as, the query "* up with" which would lead to two required answers of "come" and "keep"). Each set ({head, tail} × {collocation}) of unambiguous queries contain about 300 entries.
4. For each query, download data that matched the query from the search engine (only summaries). The maximum data amount was set at 1000 matches for each query. (There are queries with fewer available matches.)
5. Calculate the following three rates:

   Best Rate The rate of required answers appearing as the highest ranked.

   Top N Rate The rate of required answers appearing among the top N.

   Candidate Rate The rate of required answers appearing among all.

Table 3: Overall Kiwi Performance

	best (%)	top 10 (%)	candidate (%)
Kiwi (head)	**A** 77.0	93.3	**B** 97.0
Kiwi (tail)	**A** 78.5	92.8	**B** 96.3
baseline(head)	32.8	75.5	**C** 98.9
baseline(tail)	33.6	80.8	**C** 97.4

For comparison, a baseline was calculated similarly by looking at the query matches in the descending order of the search engine proposals. The rates are counted with respect to an exact match for the Kiwi results, whereas the baseline results are counted as correct when the corresponding string includes the required answer.

4.2 Performance using Collocations

Table 3 shows the Kiwi and the baseline results for collocations. The best and the top-10 rate of Kiwi outperformed the baseline. For the best rate, Kiwi records were twice as good as those for the baseline. Also, nearly 95% of the Kiwi results contained the required answers among the top 10, which was superior to the baseline results. We thus concluded that the Kiwi aggregation process was effective.

The overall error rate for a required answer that occurred at least once was:

$$(\mathbf{C} - \mathbf{B}) + (\mathbf{B} - \mathbf{A})\ \%$$

where each term corresponds to extraction and ranking error, respectively. If the required answer does not occur frequently enough on the web, Kiwi cannot obtain enough information to totalize the result. For example, for the query "be anxious *", the best candidate was considered "to" although the required answer was "for", and this case was counted as an error. Such situations occur when our required answer differs from what is the true *answer* on the web. The cause lies in how we prepare the test set, too.

The extreme case is when the required answer was not included in the downloaded data obtained from the search engine: the answer could not be processed. The error rate attributable to such cases equals:

$$(100 - \mathbf{C} \text{ of Table 3})\%.$$

As this error was not caused by Kiwi itself, the value of **C** gives the upper bound of Kiwi's performance. The potential power of Kiwi is demonstrated by how close the value **B** was to this upper bound.

5 Conclusion

Kiwi is a web-based usage consultation tool based on search engines. When the user enters a string of words to check the usage, the system sends the query to a search engine to obtain a small corpus related to the string. The corpus is then analyzed and the results are displayed.

Our system differs from existing systems in two ways. First, it is language independent, so queries can be made in any language if the mother search engine supports that language. This is achieved by string-based processing of the candidate extraction and ranking. We formalized the extraction through term branching, and ranked by candidate frequency and length. Second, flexible queries can be made using wild cards and comparisons.

According to our evaluation, the missing parts of collocations were provided by the highest ranked candidate proposed by Kiwi 70% of the time. When the top 10 candidates were considered, nearly 95% of the required answers were found. The effect of totalization was clarified by comparing the Kiwi results with the raw search engine baselines.

References

E. Brill, J. Lin, M. Banko, S. Dumais, and A. Ng. 2001. Data-intensive question answering. In *TREC*, pages 393–400.

S. Dumais, M. Banko, and Brill et al. 2002. Web question answering: Is more always better? In *SIGIR*, pages 291–298.

GoogleFight. 2000. Google fight : Make a fight with googlefight. Available at `http://www.googlefight.com/`.

F. Keller, M. Lapata, and O. Ourioupina. 2002. Using the web to overcome data sparseness. In *EMNLP*, pages 230–237.

M. Nagata, Saito T., and Suzuki K. 2001. Using the web as a bilingual dictionary. *ACL workshop DDMT*.

H. Nakagawa and T. Mori. 2002. A simple but powerful automatic termextraction method. In *Computerm2: 2nd International Workshop on Computational Terminology (COLING-Workshop)*, pages 29–35.

G. Peters. 2002. Geoff's googleduel! Available at `http://www.sfu.ca/ gpeters/cgi-bin/pear/`.

Webcorp. 1999. Webcorp home page. Available at `http://www.webcorp.org.uk/`.