用例検索システム Kiwi の知識テキストマイニングツールへの拡張

藤本 宏涼* 国安 結† 中川 裕志‡ 吉田 稔‡ 清田 陽司‡

1 はじめに

テキストマイニングには従来形態素の頻度や相関関係、構文解析などを用いるもの、またはコーパスに依存したツールが用いられてきた。しかし、コーパス中の記述表現を用いた、ユーザー対話型の手法はあまり開発されていない。本論文では用例検索システム Kiwi[1] を基に、検索クエリ周辺文字列の分岐、頻度情報を利用し、候補の切り出しと並び替えを行うテキストマイニングツール Kiwi と知識マイニングに応用した例について述べる。

2 テキストマイニングツール

2.1 概要

図 1 は本ツールの概要を表す。まず対象とするローカルなマシン上のコーパスを形態素解析し、品詞情報を付加した bi-gram インデックスの作成する。検索の手順は

1. bi-gram インデックスを用いたクエリの位置の検索
2. 品詞情報を用いたクエリ周辺形態素列の抽出
3. 形態素単位で Trie 構築、候補の切り出し、並び替え

である。形態素解析器は java で実装されている Sen を用いた。Kiwi のアルゴリズムは java で記述し、インデックスの管理には関係データベースである MySQL を用いた。java と MySQL との接続には JDBC を用いた。

2.2 データベース

インデックスの管理は MySQL を用いて行っている。使用したテーブルの ER 図を図 2 に示す。インデックスは bi-gram 単位で行い、各 bi-gram に対し、どのファイルの、何文字目に出現するか、何番目の形態素目か、その形態素の何文字目かという情報を与える。それぞれを $file\_id$、$char\_id$、$word\_id$、w_char_id とする。filelist には $file\_id$ に対するファイルの path を、data には $file\_id$ に対するファイルを形態素解析し、形態素の配列として格納する。各形態素に対する品詞

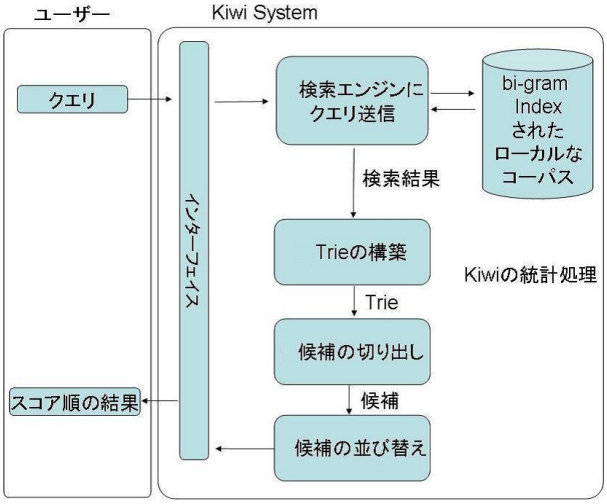

図 1: Kiwi システムの概要

情報を pos テーブルへの参照の配列として格納する。pos には pos_id に対する品詞情報を格納する。

2.3 クエリの位置検索、周辺形態素列の抽出、Trie の構築

システムはクエリを受け取ると、クエリを bi-gram に分割し、各 bi-gram のインデックスを取り出す。取り出したインデックスを参照し出現位置を検索し、その後、得られたインデックスを用いてクエリの周辺形態素列を取り出す。各 bi-gram には図 2 に示す形態素の情報が付加されているため、検索して得られたインデックスからクエリの先頭文字の位置情報と、どの形態素の何文字目かという情報がわかる。これを利用して、クエリの前方の形態素列を取り出す。取り出した、形態素列を用いて Trie を構築する。Trie の各ノードは 1 形態素に対応しており、品詞情報も付け加える。これにより例えば名詞から始まる候補だけを取り出せる。また“ 。”など記号で終わる候補は取り出さないという要求に応えることができる。前方検索時のクエリ前方形態素列の取り出し方を示す。

図 3 はクエリが“ *abcde ”の時の例である。図中の $content$ は配列であり、中身は形態素である。$word\_id = i$、$w\_char\_id = j$ とし、$j = 1$ の場合と $j \neq 1$ の場合に分け考える。

i) $j = 1$ の場合

クエリの先頭文字列は形態素の始まりであ

*東京大学 大学院情報理工学系研究科
†東京大学 大学院情報学環 学際情報学府
‡東京大学 情報基盤センター

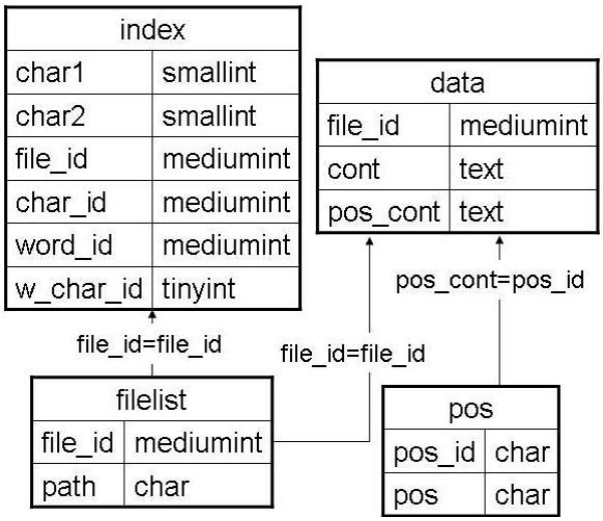

図 2: ER 図

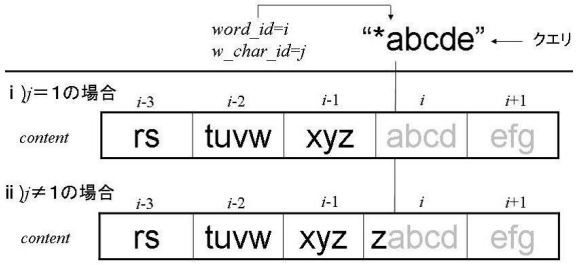

図 3: 前方検索

るので、それ以前の形態素列を取り出す。$subcont = \langle content[i-1], content[i-2], content[i-3], content[i-4], content[i-5], content[i-6] \rangle$ を返す。

ii) $j \neq 1$ の場合

クエリの先頭文字列は形態素の途中であるので、$content[i]$ からクエリの前方を取り出し、1 形態素とする。$content[i]$ のクエリの前方の部分を $last$ とすると図中での $last$ =“ z ”である。$subcont = \langle last, content[i-1], content[i-2], content[i-3], content[i-4], content[i-5] \rangle$ を返す。

2.4 検索機能

テキストマイニングのために、以下の検索機能が使用できる。

・ワイルドカード

クエリに“ * ”含まれている場合、その位置に現れる形態素列を提示する。ワイルドカードはクエリの先頭、末尾、および中間に記述することが可能であり、ワイルドカードは 2 つまで使用することができる。2 つ使用する場合は“ $*a*$ ”[1] とすることで、a の前後に現れる形態素列を提示する。

・OR 検索

クエリを“ $a(b:c)*$ ”とすることで“ $ab*$ ”、“ $ac*$ ”のどちらかの後方に出現しうる候補を検索する。

・品詞による絞込み

インデックスに形態素の情報を付加することで、品詞による絞込みが可能となった。検索する際に、品詞を指定することで“ $a*$ ”の“ * ”の部分に来る形態素の品詞を限定することができる。

・辞書の使用

辞書を指定することで、検索時にクエリ中に辞書に含まれる形態素がある場合、自動的に OR 検索する。例えば、辞書に“ 品詞 ”=“ POS ”という情報が存在し、クエリが“ 品詞の情報を* ”と与えられた場合、クエリを“ (POS:品詞) の情報を* ”と自動的に変換して OR 検索する。

3 評価

本章では各種評価実験を報告する。まず、文字ベースの Kiwi[2] と本論文で提案した形態素ベースの Kiwi での得られる結果の有効性について比較する。次に形態素ベースとした Kiwi で、評価関数に関する実験も行う。

3.1 文字ベースの Kiwi と形態素ベースでの Kiwi の比較

コーパスには (株) 日本航空インターナショナルで収集されている航空安全レポート[2]を使用する。航空安全レポートは、実際に発生したこと、またそれに対する対処について書かれたものである。このコーパスから発生した事象の因果関係を取り出すことを目的にする。このためにまず、実際に起こった事象を Kiwi を用いて検索する。得られた結果が因果関係を取り出す上で有効であるかどうかを以下の基準で評価する。以降、実際に発生した事象のことを“ Event ”とする。

1. 結果から Event に関する情報を得られるか
2. さらに検索し Event に関する情報を得られるか
3. 形態素の途中で切れているかどうか

それぞれの基準について上位 10 件、30 件を用いて基準に適するものの割合を計算する。 表 1、2 に実験結果を示す。表によると、基準 1、2 ともに 10 位、30 位以内での割合において形態素ベースの Kiwi が文字ベースの Kiwi [2] を上回った。形態素ベースの Kiwi において、一回の検索で得られる情報は 3 割程度になるが、さらに検索することで 8 割から 9 割の結果が有効に使えることがわかる。また表 2 は形態素の途中で切れている結果の割合を示したものである。文字ベースの Kiwi では途中で切れてしまう結果が 1 割近くあっ

[1]“ ”中のイタリック体で書かれた英文字は変数を表す。

[2]個人のプライバシー保護のため、個人の特定に繋がる情報は省いてある。航空安全情報は、航空の安全に資するために使用したものである。

表 1: 有効な情報の抽出

	基準	10 位 (%)	30 位 (%)
形態素ベースの Kiwi	1.	38.0	32.0
	2.	51.0	52.9
	合計	89.0	84.9
文字ベースの Kiwi	1.	27.6	25.1
	2.	41.8	41.3
	合計	69.4	66.4

表 2: 形態素の途中で切れいない結果の割合

	10 位 (%)	30 位 (%)
形態素ベースの Kiwi	0.0	1.4
文字ベースの Kiwi	8.2	9.5

たが、形態素ベースの Kiwi ではほぼなくなっているのがわかる。形態素ベースの Kiwi で途中で切れてしまう候補は、英単語、航空専門の用語が途中で切られて、出現したものであった。

3.2 形態素ベースの Kiwi の評価関数の比較

Kiwi ではスコア付けの尺度として、頻度、長さがある。現行の Kiwi では頻度× log(文字列の長さ+1) を評価関数として用いているが、形態素ベースで処理した場合、評価関数に文字列の長さを用いると、1 形態素の文字数が多いものが優先されてしまう可能性がある。そこで、文字列の長さだけではなく、形態素数を尺度として比較評価してみる。評価関数として

1. 頻度× log(文字列の長さ+1)
2. 頻度× log(形態素数+1)
3. 頻度×形態素数

を用いて比較実験を行う。それぞれの評価関数について前節で示した基準で評価する。表 3 に実験結果を示す。基準 1 について頻度×形態素が頻度× log(文字列の長さ + 1) よりわずかに上回ったのを除くと、頻度× log(文字列の長さ + 1)、頻度×形態素数、頻度× log(形態素数 + 1) の順で結果がよかったのがわかる。形態素をベースとした場合、取り出された形態素列は平均 1.84 形態素程度と短かいため、評価関数に形態素数を用いると頻度が重視されてしまう。このため、形態素数を用いると頻度の高い一般的な語句上位に来てしまいスコアが落ちたと考えられる。

表 3: 評価関数ごとの結果

	基準	10 位 (%)	30 位 (%)
頻度× log(文字列の長さ+1)	1.	38.0	32.0
	2.	51.0	52.9
	合計	89.0	84.9
頻度× log(形態素数+1)	1.	36.0	28.7
	2.	42.0	48.2
	合計	78.0	76.9
頻度×形態素数	1.	40.0	33.4
	2.	39.0	44.9
	合計	79.0	78.2

4 知識抽出の方法論

本節では知識マイニングのための方法論を述べる。航空安全レポートをコーパスとし、Kiwi を用いて、因果関係を見つけ出す方法を提案する。

因果関係を取り出すには、まず実際に発生している事象 (以下“ Event ”) を取り出す必要がある。“ 出発遅延 ”、“ 引き返し ”などが Event となる。取り出した Event をもとに、各 Event が発生する原因 (以下“ 原因 Event ”) と、原因 Event の結果発生した事象 (以下“ 結果 Event ”) を取り出す。コーパス中から原因 Event と結果 Event のペアを取り出し、各ペアを繋ぎ合わせることで因果関係を取り出すことができる。まとめると次の手順になる。

1. Event を抽出する
2. 手順 1 で求めた各 Event を結果 Event とみなし、その原因 Event を抽出しペアとする
3. 手順 1 で求めた各 Event を原因 Event とみなし、その結果 Event を抽出しペアとする
4. 手順 2、3 で取り出した原因 Event、結果 Event をグラフ構造化する

それぞれの方法について以下に詳細を示す。

4.1 Event の抽出

Event を抽出するための方法を示す。コーパス中に“ a による b ”という記述があった場合、一般的に a が原因となり b が結果を表している場合が多い。そこで、まずクエリ“ *による* ”とし Kiwi で検索し、原因 a、結果 b のペアを取り出す。次にこのペアを利用して、“ $a * b$ ”を検索することで、“ による ”以外の原因と結果の間の言語表現 c 取り出す。その後、取り出した c を用いて前方検索“ $* c$ ”、後方検索“ $c *$ ”を検索することで Event を取り出す。まとめたものを以下に示す。

アルゴリズム Event-Search:Event の抽出

入力：なし

出力:$Event = \{event_1, event_2, \cdots, event_{n-1}, event_n\}$

手順 1：原因、結果ペアの集合の抽出

Kiwi で“ *による* ”を検索する。k 番目の結果の前方を $factor_k$、後方を $result_k$ とし、そのペアを $p_k = \{factor_k, result_k\}$ とする。また得られた結果の集合を $P = \{p_1, p_2, \cdots, p_{l-1}, p_l\}$ とする。

手順 2：原因と結果の間の言語表現の抽出

各 p_k に対し、クエリ“ $factor_k * result_k$ ”

とし検索し、原因と結果の間の言語表現を検索する。得られた言語表現の集合を $Cause = \{cause_1, cause_2, \cdots, cause_{m-1}, cause_m\}$ とする。

手順 3：Event の取り出し

各 $cause$ を用いて、前方検索“ $*(cause_1 : cause_2 : \ldots : cause_{m-1} : cause_m)$ ”、後方検索“ $(cause_1 : cause_2 : \ldots : cause_{m-1} : cause_m)*$ ”を行う。両者の結果集合を $Event = \{event_1, event_2, \ldots, event_{n-1}, event_n\}$ とし、Event を返す。

4.2 各 Event の原因 Event の抽出

前述の Event-Search では、個々の Event は抽出できたが、まだ Event 間の因果性は取り出せていない。そこで、各 Event に対し、原因を抽出する方法を示す。一般的に a が原因で b が発生した場合、“ a により b が発生した ”など原因は結果の前方に書かれていることが多い。そこで、Event を結果 Event と考え、コーパス中に結果の直前にくる言語表現を検索し、その結果を用いて原因 Event を取り出す。詳細を以下に示す。

アルゴリズム Factor-Search:原因 Event の抽出**—**

入力:$Event = \{event_1, event_2, \cdots, event_{n-1}, event_n\}$

出力：ペアの集合 $P = \{p_1, p_2, \cdots, p_{m-1}, p_m\}$

手順 1：結果の直前にくる言語表現の抽出

各 $event_k$ に対し、“ $*event_k$ ”を検索する。得られた結果の集合を $Cause_k = \{cause_{k,1}, cause_{k,2}, \cdots, cause_{k,l-1}, cause_{k,l}\}$ とする。

手順 2：原因 Event の取り出し

各 $event_k$ に対し、手順 1 で抽出した $Cause_k$ を用い“ $*(cause_{k,1} : cause_{k,2} : \ldots : cause_{k,l-1} : cause_{k,l})event_k$ ”を検索する。その結果を $Factor_k = \{factor_{k,1}, factor_{k,2}, \ldots, factor_{k,i-1}, factor_{k,i}\}$ とし、各 $factor_{k,j}$ と $event_k$ をペア $p_i = \{factor_{k,i}, event_k\}$ とする。p_i の集合を P とする。

4.3 各 Event の結果 Event の抽出

Factor-Search では原因を抽出するために前方の言語表現を検索し、その結果を利用した。逆方向 (後方) を検索することで、各 Event 結果 Event を取り出せる。つまり 4.2 のアルゴリズムの手順 1 で、“ $event_j*$ ”を検索し、その結果を用い手順 2 で“ $event_j(cause_{j,1} : cause_{j,2} : \ldots : cause_{j,l-1} : cause_{j,l})*$ ”を検索し取り出す。

4.4 取り出した原因、結果のグラフ構造化

以上の方法で、コーパス中から原因と結果のペアを取り出すことができた。ペア $\{a, b\}$ とペア $\{b, c\}$ があった場合、$a \rightarrow b \rightarrow c$ という関係が取れる。これにより、c の発生を防止するためには b だけでなく a の対策をしなければいけないという知識が抽出できる。ペア同士を繋げ、グラフ化することで隠れた因果関係を取り出すことができる。図 4 は航空安全レポートを用いて上記の方法で因果関係を取り出した応用例である。ただし各課程において、Kiwi の検索結果は全ては用いていない。現在の Kiwi では結果の中にまだノイズが多い。そこで、今回は Kiwi の検索結果から人手で適切なものを取り出し使用した。この選別の完全な自動化は今後の課題である。

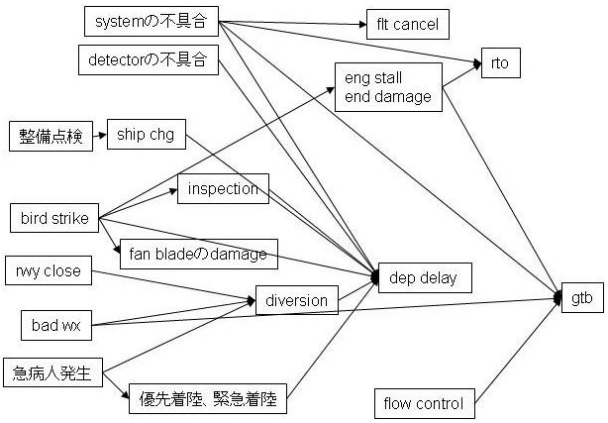

図 4: 因果関係

5 まとめと今後の課題

本論文では用例検索技術 Kiwi を知識テキストマイニングツールに拡張した。また (株) 日本航空インターナショナルにおいて収集された航空安全レポートを用いた実例を報告した。

参考文献

[1] Kumiko Tanaka-Ishii and Hiroshi Nakagawa. A multilingual usage consultation tool based on internet searching -more than a search engine, less than QA-. In *WWW Conference,* 363-371, 2005

[2] 藤本 宏涼, 吉田 稔, 清田 陽司, 中川 裕志. ローカルコーパスからのテキストマイニングツール:PortableKiwi. 言語処理学会 第 11 回年次大会発表論文集, pp. 97-100, 2005.