Wikiwi: テキストマイニングによるWikipedia検索支援

Wikiwi: A Text-Mining-Based Query Support System for Wikipedia Search

吉田 稔[*1] 中川 裕志[*1]
Minoru Yoshida Hiroshi Nakagawa

[*1]東京大学情報基盤センター
Information Technology Center, University of Tokyo

This paper describes a new query suggestion system based on text-mining. The system shows strings to given queries and strings synonymous to given queries in real time. Documents are indexed by suffix arrays, and the system dynamically performs text mining (usage mining and synonym retrieval) on the documents by using retrieval results for given queries using suffix arrays.

1. はじめに

近年、WWW 上や組織内に蓄積される電子的文書の量は増大の一途を辿り、それらの文書を人間が把握することが困難となっている。WWW 全体における文書量の増大のみならず、その中でのトピックの限定された部分集合（特定サイト内の Web 文書集合、Wikipedia、さらには企業内文書集合等）のサイズも増大し、WWW 全体と同様に、把握が困難となりつつある。

本稿では、そのような「一貫性のある大規模な文書集合」の例として、Wikipedia に着目し、Wikipedia に対するクエリ入力支援システムとして、「テキストマイニングによる検索支援システム:Wikiwi」を提案する。テキストマイニングとは、与えられたテキスト集合の中での、「言葉の使われ方」（主に、言葉に関する統計的情報）について分析するタスクである。Wikiwi では、入力されたクエリに対し、「用例抽出」「同義語抽出」という二種類のテキストマイニングをリアルタイムに行い、マイニング結果を提示する。我々のシステムは、テキストや索引をオンメモリに配置した Suffix Array 検索がベースとなっているが、Wikipedia が対象であることで、オンメモリにテキストを配置しながら、網羅性の非常に高いクエリ補完システムを実現できる。

図 1. に、Wikiwi[*1] の画面を示す。図中、A. に示されているのが検索窓であり、ユーザがここにクエリを入力すると、システムは入力内容に応じて用例や同義語を提示する。ここで用例とは、クエリの前方、後方にどのような文字列が頻繁に連接しているかを示したものであり（図中「B. 後方連接」「C. 前方連接」）、この場合、「マンション」の前方に「高層」や「分譲」、後方に「建設」や「管理」といった言葉が続き易いことがわかる。同様に、マンションの同義語・類義語として、「住宅」や「アパート」という文字列が用いられやすいということも提示する（図中、「D. 類義語・同義語」）。ユーザは、提示された用例等をクリックすることで、クエリの補完を行える。補完された結果は、そのまま Google へのクエリとして、検索に用いることができる。

データとしては、Wikipedia の 2007 年 5 月 29 日版のスナップショットから、`<text>`タグで囲まれた部分を取得した。取得したテキストはそのままコーパスとして用い、それ以上の後処理（タグの除去等）は行っていない。これは、Wikipedia

図 1: Wikiwi の実行画面

のタグやレイアウト情報が、同義語・類義語を発見するうえでの重要な文脈情報として機能するとの観察に基づいた措置である。取得されたテキストファイルは 926MB となった。実行は、AMD Opteron 248 (2.2GHz) + 13GB メモリのマシン上で行っており、パラメータ（後述）は、$F_1 = N_1 = N_2 = 1000$ と設定している。

2. 準備：Suffix Array

本稿では、コーパスを S、クエリを q と表記する。コーパス S は一つの文字列であり、入力が文書集合の場合は、これを連結したものを S とする。システムは、与えられたクエリ q に対し、連接語および同義語を発見する。

また、文字列 s の**文脈**を、「S 中で s に連接する文字列」（$s.x \in S$ 又は $x.s \in S$）と定義する。ここで . は文字列の結合であり、$x \in S$ は、x がコーパス S の部分文字列として出現することを示す。$s.x \in S$ のとき、x を s の**右文脈**、$x.s \in S$ のとき、x を s の**左文脈**と呼ぶ。

Suffix array[3] は、与えられた文字列の全ての接尾辞（へのポインタ）を辞書順に並べた配列であり、この配列への二分探索を行うことにより、効率的に（$|x|$ を文字列 x の長さとしたとき、$O(|q|log|S|)$ の計算時間で）部分文字列の位置を取得できる。この配列は、整数を 4 バイトで表現したとき $4|S|$ バイトのメモリを必要とする[*2]が、我々は、コーパス及び Suffix Array はすべてメモリ上に配置されていると仮定する。

連絡先: 吉田稔 mino@r.dl.itc.u-tokyo.ac.jp

*1 http://capacitas.r.dl.itc.u-tokyo.ac.jp:8080/ut-kiwi/

*2 Compressed Suffix Array 等、よりメモリ効率の良い実装も提案されているが、本システムでは通常の Suffix Array を使用する。

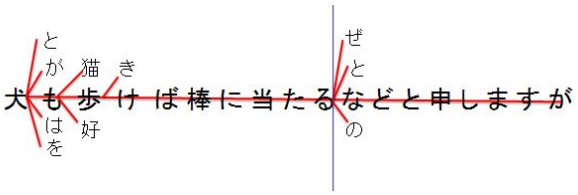

図 2: Kiwi アルゴリズム概要

Wikiwi では、$\mathcal{A}$ と $\mathcal{A}_r$ という二つの Suffix Array を使用する。前者は S から、後者はそれを反転させた $rev(S)$ から構築した Suffix Array である。$\mathcal{A}$ は右文脈の取得に、$\mathcal{A}_r$ は左文脈の取得に、それぞれ使用する。

Suffix Array により実現される操作で、実際に連接語・同義語取得において用いるものは、$nextGrams(\mathcal{A}, x)$ と $freq(\mathcal{A}, x)$ である。前者は、x を接頭辞に持つ、長さ $|x|+1$ の $\mathcal{A}$ 中の文字列を、後者は、$\mathcal{A}$ 中における x の出現回数を、それぞれ返す。(以降、明らかな場合は $\mathcal{A}$ を省略し、$nextGrams(x)$、$freq(x)$ 等と表記する。)

また、順序付きリスト $cands$ と、サイズ固定の順序付きリスト $results$[*3]を用意する。$cands$ はこれから処理される文字列を優先度の高い順に保持し、$results$ はこれまで取得された文脈や同義語候補の上位 n 個をスコアの低い順に保持する[*4]。$getFirst$ 操作を行うことにより、$cands$ からは次に処理する文字列が、$results$ からは現在**最下位**の文字列が取得されることに注意されたい。

3. 用例検索: Kiwi アルゴリズム

用例検索は、用例抽出アルゴリズム Kiwi[7] を用いて行う。Kiwi では、クエリを検索エンジンで検索し、その結果をマイニングに用いていたが、Wikiwi では、これを Suffix Array によるオンメモリの検索に置き換えることで、応答時間を大幅に短縮し、クエリ補完に利用可能な速度を実現している。

図 3. に、Kiwi アルゴリズムの概要を示す。Kiwi アルゴリズムでは、ある文字列 s が与えられたとき、「s に連接する文字の種類数」(以下、「分岐数」) を計測する。例えば、図 3. では、「犬も」の後続文字として、「猫」「歩」「好」の 3 種類の文字があり、分岐数は 3 となる。文字列 s にどのような文字列が連接するかを、木構造の探索により列挙していくが、そのさい、分岐数が増加している場合のみ、用例候補として考慮する。(図 2 の例では、例えば「犬も歩けば棒に当たる」が、分岐数 1 から 4 へ変化する点として、候補になる。) これは、「分岐数の増加は、意味的な切れ目とよく合致する」という経験則に基づいたルールである。また、Wikiwi では、用例候補が見つかったノードに対しては、それ以上深く探索を行わないという制限を加えることにより、探索の高速化を行っている。最終的に、このようにして得られた用例候補を、スコア関数

$$score(s) = freq(s) \cdot \log(1 + length(s))$$

により順位付けし、提示する。

*3 サイズ固定リストは、赤黒木を実装した Java TreeSet クラスを用いる。add 操作と、それに続く removeFirst 操作を行うことにより、上位 N 個のリストを保持するデータ構造となる。

*4 すなわち、上位 n 個のうちでスコアが最低のものが最初に配置される。

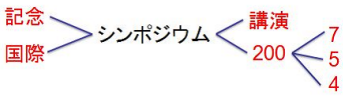

図 3: STEP-1 概要 (文字列「シンポジウム」に連接する文字列の探索木の例)

4. 同義語抽出アルゴリズム

同義語抽出において問題となるのは、ユーザーによって入力される文字列の多様性である。同義語抽出や言い換えに関しては従来より多くの研究が行われているが、その大部分は、予め規定された候補語 (例えば、「名詞」や「動詞+目的語」) どうしの類似度を測るものである。このため、本システムが想定する、「ユーザがあらゆる文字列を入力する状況」においては、文書のあらゆる部分文字列どうしについての類似度を計算する必要が生じ、必要な計算時間および記憶容量は膨大なものとなる。この問題に対し、本研究では、「ユーザからクエリが与えられた際、On-Demand にそれと類似する文字列を文書から取り出す」という問題設定を考える。このさいに要求されるのは、「どんな文字列にも対応できる」「高速な」アルゴリズムである。この実現に、Suffix Array による検索を活用する。

4.1 アルゴリズム

文字列の表記にかかわらず同義語を抽出するアルゴリズムでは、一般に、「意味の似ている語は、似た文脈で使用される」という "Distributional Hypothesis" に基づき言葉の意味類似度を測定する。特に、「隣接する文字列」は、文脈として非常に有用であることが報告されており [2]、この「隣接する文字列」は、Suffix Array により容易に検索することができる。このため、本研究では、

STEP-1 「クエリ文字列に隣接する文字列」を検索し、

STEP-2 得られた隣接文字列から適切な「文脈」を選択し、

STEP-3 「文脈に隣接する文字列」を検索する

という 3 段階の処理により同義語を抽出するアルゴリズムを開発した。長い文脈文字列を用いると高い精度を確保できる反面、取得できる同義語数が少なくなる [5]。逆に、短い文脈文字列を用いる手法では、多くの無関係な文字列が候補として取得され、精度を上げるために、対訳テキスト等の別の文脈情報が必要となる。[6] これらを勘案し、「適切な文字列」のスコアを、適度な長さの文脈を選択するよう定義する。

4.2 STEP-1,2

STEP-1,2 では、後の処理で使われる文脈の数を N_1 とし、スコア上位 N_1 個の文脈を取得する[*5]。高速化のため、文字列の検索を探索木として表現 (図 4.2・図 4.3) し、探索対象の文字列のスコア付けを探索と同時に行うアルゴリズムを採用した。これにより、枝狩りによる探索空間の削減が可能になった。

以下、簡単のため、右側連接文脈の探索についてのみ述べる。STEP-1,2 では、クエリ q に連接する文脈 x のスコアを

$$sc_c(x) = freq(q.x) \log \frac{|S|}{freq(x)}$$

*5 同一スコアの場合は、先に入った要素が高いランクとなる。

```
cands ← { '' }
while(cands ≠ ∅){
  x = getFirst(cands);
  N = nextGrams(A, q.x);
  foreach (n ∈ N){
    if (sc'_c(n) > sc_c(getFirst(results))){
      cands ← cut(n, q);
      results ← cut(n, q);
    }
  }
}
```

図 4: 文脈文字列抽出アルゴリズム

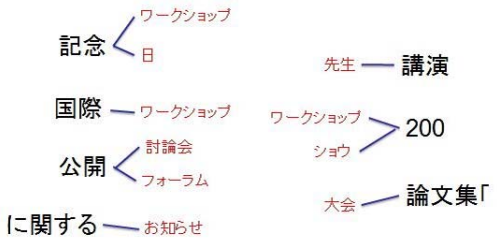

図 5: STEP-3 概要（各文脈に連接する文字列の探索木の例）

で定義し（$freq$ は頻度、$q.x$ は文字列 q と x の連結、$|S|$ はコーパス全体の文字列長)、スコア上位 N_1 個の文脈を取得するが、その際に、

$$sc'_c(x) = freq(q.x) \log |S|$$

なる上限を用いて枝狩りを行う。$\log |S| \geq \log \frac{|S|}{freq(x.y)}$ かつ $freq(q.x) \geq freq(q.x.y)$ であるため、$sc'_c(x) \geq sc_c(x.y)$ が如何なる y についても必ず成り立つ。すなわち、$sc'_c(x)$ が現在 N_1 位の候補のスコアを下回れば、それ以上の探索を打ち切ることができる。

実際には、図 4 の手順で探索が行われる。$nextGrams$ が、木構造中での子ノードの取得に相当する。ここで '' はヌル文字列（長さ 0 の文字列）であり、また、s から文脈文字列 c を取り除く操作を $cut(s, c)$ と表記している。

また、頻度があまりにも高い文脈文字列を排除するため、文脈頻度の上限パラメータ F_1 を導入し、$freq(c) > F_1$ の場合は、c を $results$ には加えない。これは、高頻度な文脈語は、一般的な語であることが多く、候補語の特定能力が弱いことと、実行時間を大きく増大させてしまうことを考慮した措置である。

4.3 STEP-3

STEP-3 では、効率的な枝狩りを行うため、

Stage-1: 左文脈のみを用い、スコア関数 $sc_l(c)$ の上位 N_2 個の候補を取得する

Stage-2: 右文脈のみを用い、スコア関数 $sc_r(c)$ の上位 N_2 個の候補を取得する

Stage-3: より精緻なスコア関数 $sc(c)$ により、絞られた候補語のリランキングを行い、上位 N_2 候補を取得する

という 3 段階の処理を行う。

```
forall(c ∈ C_l){ cands ← (c,'') }
while(cands ≠ ∅){
  D = ∅;
  while (x does not change){
    (c, x) = getFirst(cands);
    Y = nextGrams(A, c.x);
    foreach(y ∈ Y){
      y' = cut(y, c);
      D = D ∪ {(c, y')}
      sc_l(y') = sc_l(y') + 1; } }
  forall ((c, d) ∈ D){
    if (sc_l(d) > sc_l(getFirst(results))){
      cands ← (c, d);
      results ← d; } } }
```

図 6: 同義語候補抽出アルゴリズム

スコア関数 $sc_l(c)$（$sc_r(c)$）は「c の左側（右側）に連接する文脈の種類」と定義される。sc_l、sc_r 共に、探索の深さに対する非増加な関数 $(sc_l(x) \geq sc_l(x.y))$ となるため、現在のスコアが N_2 位のスコアを下回った場合、そこで探索を打ち切ることができる。

実際には、図 6 の手順で探索が行われる（ここでは、左文脈による候補語検索（Stage-1）についてのみ解説する。Stage-2 についても全く同様のアルゴリズムで行える。)。C_l は左文脈の集合である。$cands$ の要素は、ペア (c, x) であり、x の優先度でソートされている。現在のスコアが高い程優先度も高くなるよう定義する。現在最も有望な候補 x の子ノードを探索し、現在保持されている最低スコアの候補を上回る候補のみを新たに $cands$ に追加する。

こうして得られた 2N 個の候補を、最終的に、スコア関数

$$sc(x) = \sum_{c \in C_l} \log \frac{freq(c.x)}{freq_{exp}(c.x)} + \sum_{c \in C_r} \log \frac{freq(x.c)}{freq_{exp}(x.c)}$$

により順位づけする。ここで $freq_{exp}$ は、

$$freq_{exp}(x) = freq(x) \cdot \frac{freq(c)}{|S|}$$

で定義される、「c と x が無関係だった場合の x の予測頻度」をモデル化した値である。

List Cleaning 文脈リストや候補語リストは、重複した文字列を多く持つ。例えば、「研究に関する」と「に関する」といった複数の文字列が同一のリストに入っている場合がある。我々は、このような冗長性を排除するため、得られた文脈リストおよび候補語リストに対し、List Cleaning という後処理を行った。List Cleaning では、n_1 番目の要素が n_2 番目の要素の部分文字列だった場合、$\max\{n_1, n_2\}$ 番目の要素（より順位の低い要素）をリストから削除する[*6]。文脈リストの List Cleaning により、実行時間を短縮するだけでなく、得られた結果も改善することが確認された。文脈の冗長性は、スコア計算の際に重複した文脈に関する部分を多くカウントしてしまうという不均一性を生み出すため、スコアに悪影響を与えるためだと推測される。また、候補語リストに対する List Cleaning でも、結果が改善されることが確認された。

*6 List Cleaning により、実際に得られるリストのサイズが、パラメータ N_1 や N_2 よりも小さくなることに注意されたい。

5. 実験

本研究で提案する同義語抽出アルゴリズムに関し、実験を行った。実験は航空分野のレポート（7Mbytes）と、それに付随する同義語辞書を用い、「辞書に登録された語に対し、同義語を検索する」という問題設定により行った。システム実行環境は、Intel Core Solo U1300 (1.06GHz) プロセッサと 2GB メモリである。同義語辞書は、見出し語 t と、その同義語集合 $S(t)$ のペア $(t, S(t))$ の集合であり、見出し語の数は 404、平均同義語数は 1.92 であった。

クエリ t に対するシステムの出力リストを $\langle c_1, c_2, ..., c_n \rangle$ としたとき、平均精度は、

$$\frac{1}{|S(t)|} \sum_{1 \leq k \leq n} r_k \cdot precision(k),$$

と計算される。($precision(k)$ は、上位 k 個を見た時の正解率。また、r_k は、$c_k \in S(t)$（c_k が正解）なら 1、そうでなければ 0 をとる変数。）

また、パラメータは、複数の値で実験した結果、最も平均精度 (40.32) の良かった $F_1 = N_1 = N_2 = 1000$ を採用した。

Average Precision を指標として用いたところ、従来一般的に用いられる、「周辺単語」[2] と「構文構造」[4] を用いる同義語抽出手法 (ベクトル空間モデルとコサイン類似度によるランキング手法, VSM) の精度は、複数の重みづけ手法により実験したところ[*7]、最高で 57.35 ポイント（最低で 23.25 ポイント）だったのに対し、提案手法の精度は 52.20 ポイントであり（表 1）、従来手法には及ばないものの、最高精度に近い精度を得ることができた。また、1 クエリあたりの応答時間は、約 2 秒であった。

6. 関連研究

同義語抽出の先行研究には、依存構造解析を行い、単語の依存関係を文脈として用いる手法 [10] や、動詞-目的語関係を抽出し、それを文脈として用いる手法 [1][4] 等がある。また、文書外のリソースを用いる手法としては、辞書や対訳コーパスを文脈として用いる手法 [9][8] がある。これらを文脈情報として利用することにより、より精度の向上が期待できるため、これらの情報を我々の Suffix Array ベースの同義語抽出にどのように取り入れるかは今後の重要な研究課題である。

7. おわりに

テキストマイニングに基づく検索支援システム Wikiwi を提案した。Wikiwi では、クエリ入力に対し、文書集合から「用例抽出」「同義語抽出」を On-Demand に行い、ユーザに提示する。提案アルゴリズムは、任意のクエリ入力に対し用例・同義語を抽出できることが特徴である。7Mbytes のコーパスに対する同義語抽出実験では、1 クエリ当たりの応答時間は約 2 秒であり、抽出精度は、一般的に用いられるアルゴリズムと比較し、やや劣る値であった。今後は、より大規模なコーパスでの実験および高速化を図る予定である。

表 1: 候補ランキングタスクによる平均精度の比較 (%).

アルゴリズム			
提案手法	52.20		
	構文構造	周辺単語	組み合わせ
VSM (logTF)	34.43	55.61	**57.35**
VSM (TF-IDF)	24.72	37.85	39.19
VSM (TF)	23.25	40.12	40.87

*7 実際には、VSM では、与えられた候補語のランキングを行うことしかできないため、「同義語抽出」ではなく、辞書の見出し語を並び替える「候補語ランキング」のタスクでの平均精度で比較した。

参考文献

[1] C. Gasperin, P. Gamallo, A. Agustini, G. Pereira Lopes, and V. de Lima. Using syntactic contexts for measuring word similarity. In *Proceedings of the ESSLLI'01 Workshop on Semantic Knowledge Acquisition and Categorisation*, 2001.

[2] Masato Hagiwara, Yasuhiro Ogawa, and Katsuhiko Toyama. Selection of effective contextual information for automatic synonym acquisition. In *Proceedings of COLING/ACL 2006*, pages 353–360, 2006.

[3] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of the first ACM-SIAM Symposium on Discrete Algorithms*, pages 319–327, 1990.

[4] Akiko Murakami and Tetsuya Nasukawa. Term aggregation: mining synonymous expressions using personal stylistic variations. In *Proceedings of COLING-04*, 2004.

[5] Marius Pasca and Peter Dienes. Aligning needles in a haystack: Paraphrase acquisition across the web. In *Proceedings of IJCNLP-05*, pages 119–130, 2005.

[6] Mitsuo Shimohata and Eiichiro Sumita. Acquiring synonyms from monolingual comparable texts. In *Proceedings of IJCNLP-05*, pages 233–244, 2005.

[7] Kumiko Tanaka-Ishii and Hiroshi Nakagawa. A multilingual usage consultation tool based on internet searching —more than search engine, less than qa. In *Proceedings of the 14th international conference on World Wide Web (WWW 2005)*, pages 363–371, 2005.

[8] Lonneke van der Plas and Jorg Tiedemann. Finding synonyms using automatic word alignment and measures of distributional similarity. In *Proceedings of COLING/ACL 2006 Main conference poster sessions*, pages 866–873, 2006.

[9] H. Wu and M. Zhou. Optimizing synonym extraction using monolingual and bilingual resources. In *Proceedings of IWP 2003*, 2003.

[10] Kazuhide Yamamoto. Acquisition of lexical paraphrases from texts. In *Proceedings of COMPUTERM 2002*, pages 1–7, 2002.