2005－NL－166（2）
2005／3／10

グルー検出を元にした辞書を用いない英文エラーの検出

塩野谷　友隆[†]　梅　村　恭　司[†]

英文テキスト中にアルファベットで構成された非英語と呼ぶ部分が混入することがある．辞書を用いたエラーチェックではこのような非英語を検出することが難しい．ところでこのようなエラーは英文中のパターンを分断し二つのパターンを形成することが多い．パターンとパターンの区切れをグルーと定義すると，グルーはエラーであることが多いことが報告されているそこでこのような非英語を検出する方法としてグルー検出を元にしたアルゴリズムを提案し実装した．また大きな英文テキストを連結し語彙となるパターンを増やしたときの性能の推移を計測した．これらによりグルー検出によるエラー検出の可能性と，パターンの“広さ”の概念の有効性を示す．

An error detection method for English text by glue detection.

Tomotaka Shionoya[†] and Kyoji Umemura[†]

There may be non English part, which consists of alphabet, in English text. When using dictionary, it might be difficult to detect such parts. While such parts tend to divide text into two other patterns. We call this gap between patterns as ‘glue’. A previous reserch shows that ‘glue’ is more likely to be error. We think that using a detection of glue is useful to checking such errors. We have measured the goodness of detection by changing size of corpus. With these experiment, we have found two things. First, the glue is useful for the detection of error. Second, “area” of glue is effective for detecting it.

1. はじめに

英文に混入する英単語でないアルファベットの文字列を非英語と呼ぶ．非英語は機械翻訳やデータマイニング処理の精度を低下させる原因となるため，非英語を抽出する研究が進められている．もし人間がこのような非英語を見たとき，その部分は他とは異なって見える．これは人間の中に，同じアルファベットで構成された語でも非英語かどうか検出できるエラー検出アルゴリズムがあるからである．そのようなアルゴリズムを計算機上で実装することが出来ればこのような非英語を機械的にエラーとして検出することが可能になる．そのためにはその英語と非英語の差異を定式化する必要がある．我々は他の部分と比べて“稀”であることが英語と非英語を区別するものでは無いかと考え，“稀である”ということを定式化することにした．本研究では出現頻度と文字列長に基づく尺度を用いて英語と非英語を区別する方法を提案する．

† 豊橋技術科学大学
Toyohashi University of Technology

2. グ　ル　ー

2.1 グルーとは

“稀である”ということの検出のアプローチとしてグルーというものがある．稀，すなわち他に現れないというものの定義はいろいろ考えられるが，ここではパターンとパターンの隙間をグルーとして定義する．パターンとは反復して現れる文字列のことである．グルーの例を図 1 に示す．図 1 のようにエラーでパターンが分断された場合にグルーが発生する．全てのグルーがエラーではないが少なくともエラーに関する情報が凝縮されたものである．

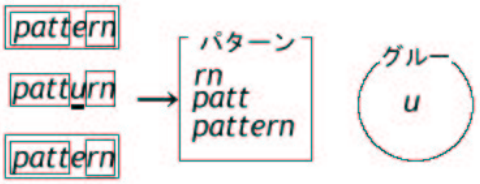

図 1　グルーの例 (矩形で囲ってある部分がパターン)
Fig. 1　An example of glue.(The part framed by square is pattern.)

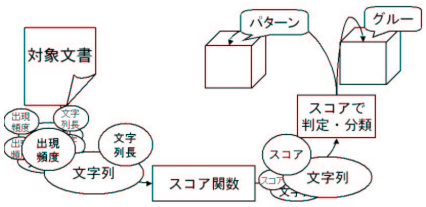

図 2 グルー検出の流れ
Fig. 2 The flow chart of detection of glue.

2.2 グルー検出の流れ

ある文字列を “パターン” と判定するためにはその文字列長と出現頻度を用いる. この二つの統計量を選択した理由は統計的なデータマイニングの定石であることと, なおかつ “ある文字列が複数回現れる” というパターンの定義に基づいた尺度であるからである. グルーを検出することがスペルミスなどの検出に役立つことが報告されており, ここでも同様に文字列長, 出現頻度を元にグルー検出を実装している.[2]. グルー検出は

- 全ての部分文字列の文字列長, 出現頻度を計算する.
- その二つの統計量を入力とするスコア関数によりその文字列にスコアをつける.
- スコアが閾値より低い文字列をグルーとする.

という流れで実装される. これを図 2 に示す.

ところで本研究では単語自体のスペルミスではなく, 非英語の部分を検出することを目的としている. そこでグルー検出を単語単位で行うように拡張した.

2.3 スコア関数

プログラム言語などの形式言語では出現頻度, 文字列長がそれぞれ大きいパターンを形成することが多いが, 英文のような文脈自由言語では形式言語ほど大きなパターンが形成されることが少ない. 従来のグルー検出では文字列の長さ, 大きさに関して式 (1) のような線形のスコアをつけていた.

$$BScore(s, L, F) = \sum_{l=1}^{l=L} \sum_{f=1}^{f=F} S(s) \qquad (1)$$

$$S(s) = \begin{cases} 1 & (L(s) > l \&\& F(s) > f) \\ 0 & otherwise \end{cases}$$

s : 文字列

ただし $L(s)$ は文字列の長さを $F(s)$ は文字列の出現頻度を返す関数, L, F はそれぞれ文字列長, 出現頻度の閾値を決定するパラメータである. スコアの大きな文字列ほど “パターンらしい” と判定される. したがって閾値を設定し, その閾値より大きなスコアを持つものをパターンとすることでグルー検出を実装するものである. このスコア関数は文字列をグルーとみなす文字列長, 出現頻度の閾値の設定で最高スコアを決めることができる. そのため最高スコアと閾値の割合を測定することで経験的なスコアの閾値を見つけることができると考えた. しかし大きなパターンが形成されにくい場合, パターンに大きなスコアが与えられない. このためスコアの閾値の決定が難しいことが分かった. そこで大きい文字列には大きなスコアがあたるように文字列長と頻度の積をスコア関数とした. このように文字列のスコアとして出現頻度と長さの関数を適用した例が他にも報告されている.[1]

$$AScore(s) = L(s) * F(s) \qquad (2)$$

これは文字列の “広さ” として考えることができる. そこでこのスコアのつけ方を AreaScoring と呼ぶ. また前述した閾値を用いるものを BoundaryScoring と呼ぶ. 本研究では BoundaryScoring をベースラインとし, AreaScoring との精度を比較することで AreaScoring の有効性を計測した.

3. アルゴリズムの詳細

3.1 SuffixArray

長い文字列の処理にあたり, 幾つかのデータ構造が用いられている[4]. 本研究でグルーを検出するにあたり我々は SuffixArray というデータ構造を用いた. これは実装が比較的容易であり, また後述する LCP と組み合わせることでグルー検出に便利だからである. SuffixArray とは文字列のある位置から末尾までの部分文字列, 接尾辞 (Suffix) を辞書順にソートしたものである. banality_banana という文字列について SuffixArray を構築したものを図 3 に示す. SuffixArray におけるパターンを図 4 に示す. SuffixArray 上の矩形の横方向の長さがパターンの長さ:length, 縦方向の長さがパターンの出現頻度:tf(term frequency) に対応している.

ここでこの SuffixArray において i 番目と i+1 番目の Suffix との LCP を計算する. LCP(Longest Common Prefix) とは二つの文字列の先頭からの一致文字数のことである. ただし, 最後の LCP は 0 と定義する. このとき, 矩形の横方向の長さとこの LCP が対応する. つまり LCP を検査することでテキストのパターンを検出できる.

3.2 SuffixArray の単語単位への拡張

本研究では文字単位ではなく単語単位でのエラー検出を目的としている. そこでこの SuffixArray を単語単位に拡張した. これを WBSA(WordBaseSuffixArray) と呼ぶことにする. SuffixArray の単語単位への拡張

index	suffix
8	_banana
14	a
3	ality_banana
12	ana
1	anality_banana
10	anana
0	banality_banana
9	banana
5	ity_banana
4	lity_banana
13	na
2	nality_banana
11	nana
6	ty_banana
7	y_banana

図 3 “banality_banana” の SuffixArray
Fig. 3 An example of Suffix Array.(structed phrase is “banality_banana”)

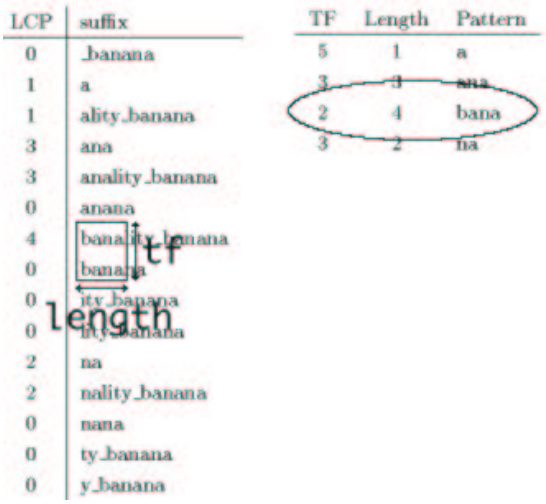

図 4 SuffixArray 及び LCP とパターンの対応
Fig. 4 The pattern compliantanted on Suffix Array and LCP.

はさほど困難ではない. 何故ならば単語の先頭さえ計算することが出来れば, 各単語は文字単位での各文字に対応する. それを図 5 に示す. 各単語と各文字がこのように 1:1 で対応するのならば, 従来の SuffixArray の構築アルゴリズムを少し修正することで実装することができる. 単語の先頭を判断するために, 単語の区切りとなる文 (デリミタ) を定義する必要がある. 本研究ではスペース, カンマ, ピリオド, カッコなど直感的なものをいくつか集合として定義した.

character based	word based
b	red
a	apple
n	light
a	apple
l	pink
i	peach
t	purple
y	grape
-	crimson
b	red
a	apple
n	light
a	apple
n	light
a	apple

図 5 文字単位から単語単位への拡張
Fig. 5 An extension from letter based to word based.

3.3 WBSA の構築

(1) 入力文字列の全ての “word” の先頭 index を計算する. ここではある word から文末までの接尾辞を WordBaseSuffix と呼ぶ.
(2) 入力文字列の全ての WordBaseSuffix について辞書順にソートを行う. 図 3 では全ての Suffix を並べたように説明しているが, 実際は先頭の index だけを記憶すればよい. したがって WBSA の構築に必要なメモリ量は入力文字列の長さ n に比例する.

3.4 LCP の計算

(1) 配列 index の長さと等しい長さの配列 lcp を用意する. この長さを m とする. lcp[i] は WBSA において i 番目と (i+1) 番目の WordBaseSuffix の先頭から一致する word 数を表す.
(2) i=0 とする
(3) WBSA の i 番目と (i+1) 番目の先頭からの一致 word 数を lcp[i] に格納する.
(4) $i < m - 1$ ならば (2) に
(5) lcp[m-1]=0 とする
(6) 終了

3.5 ステミング

データマイニングにおいてステミングという前処理を行うことがしばしばある. ステミングとは英単語の語尾を無視することで時制, および単複などによる英単語の語尾変化を吸収する手法である. 本研究においてステミングを施すことで精度に効果があるか調べる

ためにステミングを施す場合, 施さない場合の二つのアルゴリズムを比較した.

3.6 スコアの計算

3.6.1 AreaScoring

(1) 長さ m の配列 score を用意する
(2) i=0 とする
(3) $lcp[i] \neq 0$ ならば j=i+1 とする.lcp[i]=0 ならば (5) へ
(4) $lcp[i] \leq lcp[j]$ もしくは $m > j$ の間 j=j+1 とする
  (a) z=i とする
  (b)
    - $score[z] < (j-i) * lcp[i]$ ならば $score[z] = (j-i) * lcp[i]$
    - $score[z] > (j-i) * lcp[i]$ ならば $score[z] = score[z]$
  (c) $z \neq j$ ならば z=z+1 として (b) へ
(5) $i < m$ ならば i=i+1 として (4) へ
(6) 終了

3.6.2 BoundaryScoring

(1) 長さ m の配列 score を用意し,0 で初期化する
(2) width=MINWIDTH とする
(3) height=MINHEIGHT とする
(4) i=0 とする
(5) $lcp[i] \leq width$ ならば j=i+1 とする.lcp[i]=0 ならば (5) へ
(6) $lcp[i] \leq lcp[j]$ もしくは $m > j$ の間 j=j+1 とする
(7) $j - i \leq height$ ならば
  (a) z=i とする
  (b) score[z] が現在の width,height で変更されていないなら,score[z]=score[z]+1 とする
  (c) $z < j$ なら $z = z + 1$ として (b) へ
(8) i=j とする
(9) i=m ならば height=height+1 とする
(10) $height \leq MAXHEIGHT$ ならば (4) へ
(11) width=width+1 とする
(12) $width \leq MAXWIDTH$ ならば (3) へ
(13) 終了

MINWIDTH, MINHEIGHT, MAXWIDTH, MAXHEIGHT はそれぞれ BoundaryScoring におけるパラメータで文字列の幅と高さをそれぞれどのくらい重視するかというものである.

4. 評　　価

評価にあたり次のデータを用意した

> I can chcp 437 email you copies of these articles if you can't find them at yourlibrary. I've been using Managing Your Money for several years, and I have several friends who use Quicken, though I've not used it myself. CONFIG.SYS My overall impression is that Quicken is a financial accounts manager

図 6 新聞データに非英語を挿入したテストデータの一部 (下線部が検出したい "エラー")

Fig. 6 The sample of test data which is newspapers inserted noises.(The underlined words are "error" which should be detected.)

(1) 非英語データ
(2) WindowsOS に関するメール
(3) 新聞データ

ここでエラーを検出するということを非英語データを抽出することと定義する. エラーの検出は辞書を必要とせずエラー検出を施したい文字列のみで行うことができる. しかしながら短い文字列では語数が少ない上, プログラムのように少ない語彙で表現もされていない. そこで大きな英文テキストを付加し語数を増やすことで精度向上を図った. 本実験ではまずメールに非英語を挿入し, テストデータ (図 6) を構築する. これに新聞のデータを英語のパターンとして連結したものに対してエラー検出を行った. メールと新聞のデータには直感的に見て関連性が薄い. そのため意図的にエラーでない語のスコアを上昇させる効果はなく単純に学習用データとして用いても差し支えないと考えた. このように異種のデータセットをコーパスとするアプローチは 3) でも用いられており, これによって検索精度を向上させた報告がある.

4.1 関数による効果

単語比較関数とスコアリング関数をそれぞれ組み合わせて 4 つのアルゴリズムでエラーを検出した. アルゴリズムの評価には再現率 (3) と適合率 (4), 及び F-尺度 (5) を用いた.

- 再現率 (recall):完全性を評価するための尺度. 検索漏れの少なさを示す.
- 適合率 (precision):正確性を評価するための尺度. 検索ノイズの少なさを示す.

$$\text{再現率 } (recall) = \frac{\text{検出された非英語数}}{\text{全文書中の非英語数}} \quad (3)$$

$$\text{適合率 } (precision) = \frac{\text{検出された非英語数}}{\text{検出された単語数}} \quad (4)$$

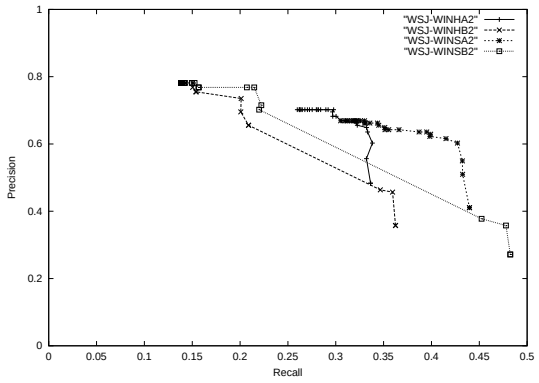

図 7 再現率-適合率グラフ
Fig. 7 Precision and recall.

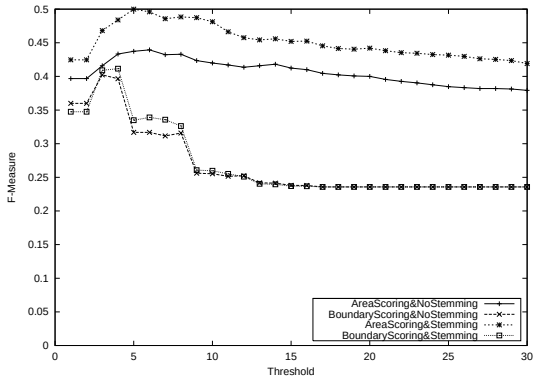

図 8 Score の閾値に対する F-尺度
Fig. 8 F-measure and threshold of score.

一般的に両者はトレードオフの関係にある. パラメータを変えることで再現率・適合率を変化させ, それをプロットしたグラフが上にある検出アルゴリズムが優れているとされる. 各アルゴリズムの再現率-適合率をプロットしたものを図 7 に示す. AreaScoring を施したものが BoundaryScoring を施したものよりも上に来ている. またステミングを施すと性能がよくなることも分かる.
各アルゴリズムのスコアの閾値の設定による精度の変化を F-尺度を用いて計測した.

- F-尺度:再現率Rと適合率Pの調和平均. 両者が大きな値を取るとき大きい値を取る.

$$F = \frac{2}{1/R + 1/P} \tag{5}$$

図 8 にスコアの閾値に対する F-尺度をプロットしたものを示す. このグラフから分かるように AreaScoring が BoundaryScoring より優れているといえる. また

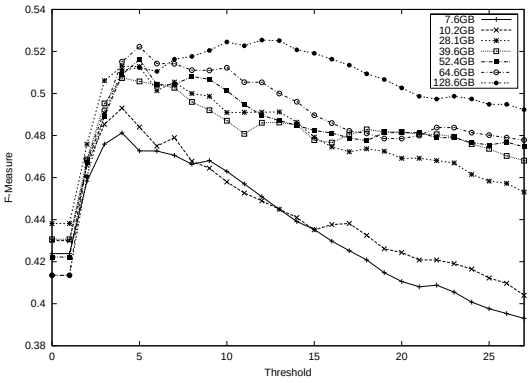

図 9 コーパスのサイズの変化に対する F-尺度の推移
Fig. 9 F-Measure fo various corpus size.

BoundaryScoring は局所的には良い精度を示すが, 変動が激しく, 調整がしにくいと言える.

4.2 コーパスのサイズに対する効果

付加するコーパスサイズを変化させ検出精度の変化を測定した. 図 9 にコーパスのサイズを変化させたときの F-尺度の推移を示す. アルゴリズムは最も精度の高かったステミングを施した上で AreaScoring を用いたものを選択した. コーパスのサイズが大きくなるにつれて F-尺度が上昇した. また, スコアの閾値の変化に対する F-尺度の変化が緩やかになりシステムとして安定したといえる.

5. 考　　察

アルゴリズムとしてステミングを施した場合の AreaScoring の使用が有効であった. このことからステミング,AreaScoring の両方がグルー検出に有効であるといえる. AreaScoring では大きなパターンには非常に大きなスコアがつくため, 一般語の除去に非常に有用である. またステミングによって時制, 単数複数の差を吸収できたことも構文や文脈による違いを吸収できたと言える.

語彙の元となるコーパスのサイズを大きくすることは, 精度を上げるだけでなくシステム自体を安定させることができた. これは語彙が少ない頃は大きなパターンとして検出できなかった慣用句や熟語などの複数の語からなる一般的なパターンが大きなスコアを得て除去できたことによると考えられる.

6. ま と め

グルー検出アルゴリズムを単語単位で実装し, それを用いたエラー検出アルゴリズムを実装した. このアルゴリズムにおいて単語の比較にはステミングを施し,

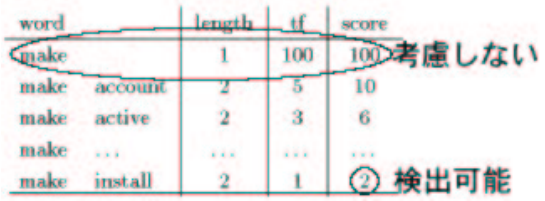

図 10 頻出語を含む非英語の検出
Fig. 10 An error detection which include common word.

スコア関数にはパターンの “広さ” を用いることが検出精度の向上に繋がることを示した. また語彙を増やすためにコーパスを付加してエラー検出を行い, コーパスのサイズに応じて精度が向上し, システムとして安定することを示した.

7. 今後の課題と展望

まず課題として挙げられるのが精度の向上である. 本アルゴリズムでは *make* や *have* などの頻出語が AreaScoring において $L(s)*F(s) = (1*LargeNumber)$ となり大きなスコアが当てられる. このため検出したい非英語に例えば *make install* のように頻出語が混じっているとき *make* を検出できない. これを解決するために, 図 10 のような短い length で大きな frequency を持つ “縦に細長い” パターンはその時点では Score を計算せずに, 次の length の段階で Score を別々に計算, すなわち式 6 のように AreaScoring の計算式を変更することが考えられる.

$$AScore(s, L, F) = \begin{cases} 0 & L(s) < L \&\& F(s) > F \\ L(s) * F(s) & otherwise \end{cases} \quad (6)$$

このことで頻出語を含む複数語で構成される非英語を検出できるようになると考えられる.

また *a* や *the* などの一般的な stop word を考慮することで図 11 に示す例のように大きなパターンを切り出すようにすることが考えられる. これらの工夫で統計的解析でのエラー検出精度は向上すると考えられる.

sentence

I have a pen
I have pens
I have the pen

Pattern	(score)	with Stop Words	(score)
I have	(2*3)	I have * pen*	(4*3)
pen*	(1*3)		
a	(1*1)		
the	(1*1)		

図 11 Stop word を考慮した時のパターンの切り出し
Fig. 11 Patterns with stop word

参 考 文 献

1) 湯元紘彰, 森辰則, 中川浩志. 出現頻度と和接頻度に基づく専門用語抽出. **情報処理学会研究報告**, *2001-NL-145*, 2001:111–118, 9 2001.
2) 梅村恭司, 吉川裕之, 貴島寿郎. n-gram 解析手法を応用したプログラムの欠損の検出. **情報処理学会論文誌**, 39:3294–3303, 12 1998.
3) 大井洋子, 大田佳宏, 今一修, 丹羽芳樹, 久光徹. 一般語とのあいまい性を持つたんぱく質名の自動検出. **情報処理学会研究報告**, *2004-NL-163*, 2004:21–28, 9 2004.
4) 坪井裕太. 頻出部分文字列のマイニング. **情報処理学会研究報告**, *2003-NL-158*.