人工知能学会 インタラクティブ
情報アクセスと可視化マイニング研究会(第1回)
SIG-AM-01-02

主体的な自己探求のためのテキストマイニングツールと web スクレイピングのためのフレームワーク

The text-mining tool for an active self-quest and the framework for web scraping

伊藤貴一、[1]　熊坂賢次 [2]

Takaichi Ito[1],Kenji Kumasaka [2]

[1] 慶応義塾大学院政策・メディア研究科
[1] Graduate School of Media and Governance, Keio University
[2] 慶応義塾大学環境情報学部
[2] Faculty of Environment and Information Studies, Keio University

Abstract: This paper describes two tools. One is making the network of the relation of language by oneself, and it is a tool which visualizes the self-recognition which nestled up to data rather than analysis of objective text data. Another is a framework for the data acquisition from a web. Although it is already known that web has a lot of data, acquiring it needs special skill. It is facilitated. For text mining, since it is a required tool, it states.

0.はじめに

この論文では二つのツールについて述べる。一つは、自分自身で言葉の関係のネットワークを作ることで、客観的なテキストデータの分析というよりも、データに寄り添った自己認識を可視化するツールである。もうひとつは、ウェブからのデータ取得のためのフレームワークである。web に大量のデータがあることは既に知られているが、それを取得するのは専門の技能を必要とする。それを簡便化するものである。テキストマイニングのためには、必要なツールであるので述べる。

1.主体的な自己探求のためのテキストマイニングツール

現在多くのテキストマイニングツールが開発されているが、それらのツールを利用するユーザは、ツールが解析した客観的で絶対的な結果を前提に、その結果を懸命に解釈する受動的な他者でしかない。そのため、テキストマイニングツールを魔法の道具だと思って、その結果をただ受け入れるという考えない人たちを作りだしてしまう。しかし、テキストには文脈や背景知識といった暗黙知が含まれており、機械的には分析しにくいものを多分に含んでいるため、考えずただ受動的にその結果を受け取るという態度は望ましくない。むしろ、ユーザは解読する主体として自らの問題意識に従って、ツールが合理的に判断した素案と対話しながら自分なりに納得する成果を導き出す、という「自己探求的で対話的な関与」を可能にするツール開発が必要とされる。このようなコンセプトに基づいて開発した、柔らかい構造化ツール『Hipparu-McS：ヒッパルーマックス）』[1]である。

そのため、テキストの客観的な事実を可視化するというよりも、客観的事実と主観的実感をすり合わせていく作業をするためのツールであるといえ、むしろ自己認識を可視化するという性質をもつ。分析者の頭を働かせるようにすることがこのツールの目的である。

また、テキストには潜在的に、いつ、どこで、だれが、書いたのかという５W1H のような情報は付随する。例えば、男が書いた文章なのか、女が書いた文章なのかという情報は、分析の手かがりになる。他にも、テキスト全体が肯定的か、否定的かという情報や、テキストの分類の結果、文章に付随する画像の情報もあるだろう。これらの情報をテキストの中に入れて分析すればいいという小技が存在するが、性質が違うものを混ぜるのはよくないので別処理した方がよい。それら文章に付随する情報を扱うための仕組みがあると分析に深みがますので、その仕組みも実装した。

2.実装

2.1 基本機能

Hipparu-McS は，手軽に使ってもらうために，特殊な形式のデータファイルを作る必要がない，ただのテキストデータを読み込むだけで，処理をしてくれるように実装した（Fig.1）。ツールの画面領域はメニューバー，サイドバー，キャンバス，データビュー，単語ランキングの 5 領域からなる（Fig.1-A）。メニューバーは入力データの指定や SAVE/LOAD 機能，サイドバーはノードの削除や表示の調整などの機能，キャンバスはノードとリンクを描く場所，データビューは文章データを見る場所，単語ランキングは注目単語を指定する場所である。

以下，挙動プロセスを示す。なお，挙動プロセスの番号は Fig.1-B 以降に対応する。

(1) メニューバーの「入力データ」「テキストファイル」を選択し，単語の頻度ランキングを作る。さらに自身の問題意識に基づいて，ランキングから適切な単語を選択し，キャンバスにノードをおく。初期状態が真っ白なのは真っ白な気持ちでデータに向

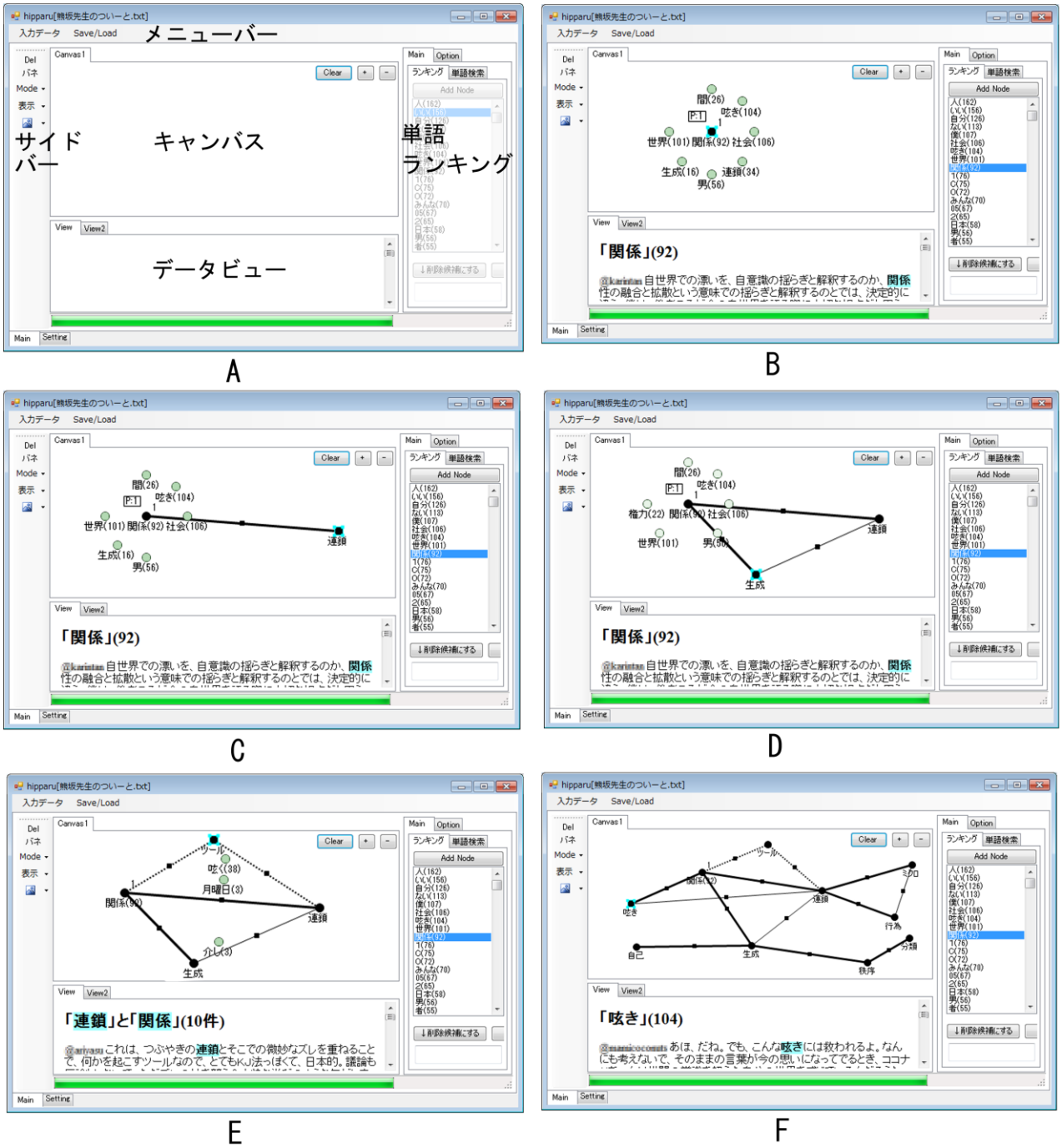

Figure 1　Hipparu-McS の挙動プロセス

かい合ってほしいためである。ユーザ自身で言葉のネットワークを探求するために，ノードはマウスで自由に動かせるようにしてある。そしてノードを選択したとき，その周辺に 7 個の関係の強い語が時計回りで表示され，データビューにその単語を含む文章が表示される。関係の強さは Jaccard（Jaccard= |A∩B|/|A∪B|）指標を使い，基本設定では一行分の文章での単語の関係を計算している。単語の後にある数字は単語の頻度である。また 7 個だけの表示では探求する単語にたどり着けないことがあるので,「P」のところをクリックすると次の候補 7 つを出すことができる。（Fig.1-B）。

(2) 候補語の中で適切だと思った単語をマウスでつかんで「引っ張る」とノードとして採用され，黒リンク（図上では太線）が作られる（Fig.1-C）。この新たに引っ張られたノードを選択すると，その周囲に関係の強い語が表示され，同様に引っ張ることができる。

(3) ネットワークを探求する過程で，すでにノードとした単語と関係の強い単語を選択したとき，新たに赤リンク（図上では細線）が自動的に引かれる。黒リンクが自己探求の線であるのに対して，赤リンクはツールが合理的判断に基づいて引いた線である（Fig.1-D）。

(4)リンク上にある黒いボタンを選択すると，リンクで結ばれる 2 つのノード単語と関係が強い単語の頻度ランキングの上位 4 つが表示される。この 4 つの候補ノードも選択すると緑リンク（図上では点線）が引かれる。この単語は，頻度は低いが，ユーザが探求したリンクと両ノードに共通する単語なので，さらに深い探求を誘発する機能をもつ（Fig.1-E）。

(5) 上記の探求行為の繰り返しによって，複雑で多様な構造図が作成される。（Fig.1-F）。

2.2 文章の属性処理

文章に属性をつけることを考えた時、理想的な形式は、XML のような構造化された形式であろう。しかし、先に述べたように、簡単に使ってもらうために、シンプルなテキストを入力データとしてもらうことを目指している。そのため、簡易的な記法を作った。

テキスト<分類カテゴリ:属性, 分類カテゴリ 1:属性 2,・・・>

このような形で、文章の後に<>で括ったものを付け加えることで認識されるようにした。＜＞の中は、カンマ区切りで、属性の数は可変である。また分類カテゴリをつけることで、属性の関係を定義する。例えば、ただの「男」「女」だけの情報では男と女に排他関係があることが機械にはわからない。そこで、「性別:男」「性別:女」とすることで他の多数の属性があったとしてもその関係がわかるようになる。

そして、属性のデータは、分析時に文章の生データを見るとき読めるようにすることや、その集計、また、にこの属性によるフィルタ機能を使えるようした。

3.分析事例

女子学生が自分の食べ歩きブログ 98 記事を素材に，Hipparu-McS を利用して，彼女自身の食意識を探求した柔らかい構造化の成果を取り上げる[2]。

おいしいという言葉に注目してグラフを作っていく中で、漠然としていた自分の食に対する意識が構造化されていった（Fig2）。その中で、自身が注目しているのは、味、香り、食感の三要素であるということに気付いた。そして、さらに、その「味」という言葉に注目することで、さらなる食に対しての変数を気づくことができた（Fig3）。自身が書いた文章をツールを通じて分析することで、自分でも思っていなかった関係性や構造が発見され,「自分はそんなことを考えていたのか」という新鮮な気づきがもたらされた。

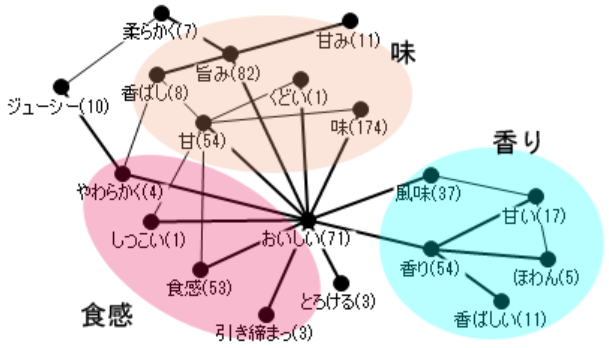

Fig. 2 「おいしい」の単語と共起関係

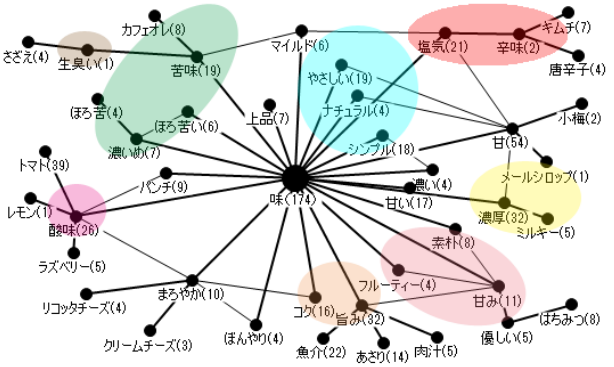

Fig. 3 「おいしい」のさらなる探究

4. Hipparu-McS の課題

このツールでは、同じテキストを分析しても、分析者が違えば、分析者のその時の関心が違えば、作られてくるグラフは全く違うものになってしまう。データに沿って作られているのだから、これをいけないことと否定的に見る必要はない。そもそもテキストを分析する視点は多様であり、そしてその解釈も多様である。しかし解釈は多様だ、というだけで終わっては単なる相対主義（主観主義）であり、その多様性を統合する視点をいかに仕組みとして組み込むか、という重要な問題が残る。そのため、同一のデータから作られる多様なグラフを統合する仕組みが必要である。また、主観重視といっても、客観的に提供される情報が貧弱であるという課題もある。今は単語のランキングと単純な単語の２者関係でしかなく、全体像として客観性というものを認識できる情報が弱い。その点も強化しなくてはならない点だろう。

あと、属性を分析するための仕組みを用意しているが、その機能もただのビューワとフィルタでしかなく、貧弱である。違う属性との比較や、同じグラフでも、属性を変えるとリンクが変化していく様子を表現する機能が必要だと考えている。

5.Web スクレイピングのためのフレームワーク

１９９０年代半ばの商用インターネット開始から、多くのサイトが立ち上がり、web 上に情報が整理され、大量の情報が置かれることになった。Web 上に大量に情報があり、それを分析したいといっても、アクセスするには、プログラムをできる人には、web アクセスのプログラミングを組めばいいのだが、一般の人では、一つ一つ web ページをブラウザでみるなどの方法しかない。大量のデータを扱うことのできるためには、個々人がプログラミングスキルを身につけるべきというは、賛同したいことであるが、現実には習得しなくてはいけないスキルが増えている現代では難しい。そのため、欲しい web 上のデータの大量取得（Web スクレイピング）を容易にするためのツールが必要である。

しかし、web 上の特定の HTML タグを取ってくるツール（例えば、掲示板の書き込み部分のみを抽出する）というのは簡単に作れるが、欲しいのは構造化されたデータ（例えば、掲示板の書き込み、書いた日付、書いた人の名前などの情報が一つの塊としてあるデータ）であり、web の多種多様性に適応するような仕組みでなくてはならない。そのために、機能を小分けし、モジュール化して、それをユーザが対象の web サイトに合わせて組み合わせるという方法がいい。そのため、ツールというよりも、フレームワークという形にした。

プログラマの立場からすると、Web スクレイピングは、今、Web 関係のライブラリはメジャーなプログラミング言語には揃っているので、自分の習得した言語で、容易に作れるようになってきている。しかし、それでもプログラミング言語のもつ決まりごと（変数の宣言、変数の一時格納、ループ処理、後処理など）に多くの労力を削がれ、書くコード量が増え、重要なパラメータがあちこちに散らばり、見通しが悪くなってしまう。そのため、必要なパラメータだけを記述して、機能を組み合わせるだけのフレームワークにするのは、全体の見通しがよくなり有益である。

機能を組み合わせるというのは、データフローを作るということに他ならず、web からのデータ取得をデータフロープログラミングするためのフレームワークでもある。これは、プログラミング言語のパラダイムで言う、手続き型言語から関数型言語に変形させることであるともいえる[3]。

このような変化は、web スクレイピングは対象のウェブサービスに依存し、そのデザインが変更された時に作り直さないといけないものなので、後に変更されることを前提とし、見通しのよさを作るのは必要なことである。

このフレームワークは、http://rawler.codeplex.com/ にて Rawler フレームワークとしてオープンソースで公開している。

似たような web スクレイピングのためのツールとして Java で作成されたオープンソースプロジェクトの Web-Harvest[4]がある。これとの違いは、Web-Harvest は XML でのプログラミングに念頭が置かれているが、Rawler フレームワークは、ビジュアルプログラミングしようという構想があったことに起因する関数だけで記述しようとしていることであると思われる。

6.仕様と実装

実装は C#で行い、使うときには XAML で記述できるようにした。XAML とは、Extensible Application Markup Language の略で、マイクロソフトの XML をベースとした拡張である。特徴としては、XML のタグが C#のクラス名であり、記述することでインスタンスが生成されるため、オブジェクトの状態と関

係を記述できる仕組みである。主にアプリケーションの外観のデザイン（ボタンの配置など）に使われている（例：WPF／Silverlight)。XAML はあくまでテキストなので、そのルールがわかっていれば、コピーアンドペーストに優れ、別のところで書いたコードがそのまま使える。また、エディタの機能で折りたたみができ、そうすると見通しがよいものになる。そのため採用した。

6.1 基底クラスの設計

XAML の仕組みを使い記述するために、すべての機能は、RawlerBase という基底クラスを継承したものになっており、オブジェクト指向言語のポリモーフィズム(多態性)を利用したものになっている。RawlerBase には、主にオブジェクトの親子関係を格納する Parent,Childern プロパティと、オブジェクト自身が持つ Text プロパティ、そして Run()メソッドがある。XAML は XML なので木構造であり、タグの入れ子関係で記述できるように親子関係の情報を持っている。Run()メソッド実行すると親の Text プロパティを参照して Run を実行し自身の Text を作り、そして子の Run()を実行するというものになっている。

そのため、XAML で作られた親子関係の木構造に沿って、深さ優先探索のように実行されていく。

木構造でのデータの流れの記述は時として、深い階層になってしまい、可読性を落とすことにつながる。例えば、部分の抽出→タグの消去→改行の削除→空白削除といった処理などをすると一気に階層が深くなってしまう。そのため、RawlerBase には PreTree プロパティがあり、そこでは、RawlerBase で前処理を記述することができ、それは、その命令が行われる前に実行される。行数を必要とするので長くなるのだが､多くの XML のツールでは折りたたみができるので、折りたたんでしまえば、苦にならない。そして、コードの見通しはよくなる。

6.2 複数処理（繰り返し処理）

HTML での Link の取得のように複数になるものもある。上に書いた方法では単数しか処理ができないので、RawlerBase を継承した RawlerMultiBase クラスをつくり子をリストに入っているデータの数だけ複数回実行する、複数処理に対応させた。このようにすることで単数複数を気にせず配置できるようになっている。命名規則として単複を意識させるため、主要な複数のものは複数形にしている。(例 Links、Tags、ReadLines など)

また、繰り返し処理の制御のためのクエリ機能の RawlerQuery プロパティがある。これは C#の LINQ (Language Integrated Query： 統合言語クエリ)に影響を受けたもので、その部分的なラッパーである。これを使うことで、初めの要素の抽出、最後の要素の抽出や、指定した条件に適合するものだけを抽出するなどの得られた複数のテキストに対する処理ができる。

6.3 継承されたクラス群

個々の機能は、RawlerBase、RawlerMultiBase を継承したものである｡Page クラスは親テキストを URL として web ページにアクセスする。Tags クラスは html を解釈して指定したタグを抽出する。Links クラスはリンクを抽出する。このような形で web ページからのデータ取得はできる。データの処理としては、Data クラスはデータを蓄積し、DataWrite クラスは Data クラスに属性を付けて Text を書き込み、NextDataRow クラスで今まで書いたのを一つの塊として確定させる。この一連の処理で構造化されたデータの取得が可能になる。

また、制御として、IF 文相当の Contains（指定した文字列が含まれていると実行される）や Equal（指定した文字列と同じ場合に実行される）Switch 文相当の Switch クラスがある。これで、構造化プログラミングの順次、反復、分岐の三要素をすべて満たすことになる。順次は XAML の木構造、反復は RawlerMultiBase クラス、分岐は、RawlerQuery や Contains クラスなどの命令群である。

そのほか、３０近くのクラスがある。ログインが必要なページにアクセスするためのログイン機能も存在する。これらを使うことで様々な種類の web ページからデータ取得だけではなく、ファイルの読み書き、さまざまな繰り返し処理、テキストの変形、エラー報告までできるものとなった。

6.4 拡張性

C#のプロジェクト内で使えば、WPF アプリの作成のようにコードビハインドとして、オブジェクトに対して追加のイベント処理ができるため、データベースとの連携処理や、さらに細かい処理をすることができる。

また、RawlerBase を継承したクラスを作れば、このフレームワークに乗っかることができる。テキストを変数として、テキスト（単数複数）を返す関数はいかようにも作れる。著者自身、再利用する可能性が高い必要な処理はその都度作っている。

他にも色々な可能性を考えることができる。たとえば、グーグルやツイッター、フェイスブックなどの WebAPI を使いデータを取得することも、継承したクラスを作ればいい。形態素分析を行うことや、

テキストの分類し、そのクラス名を返すこと、テキストがポジティブかネガティブか判定するといったテキスト分析も継承したクラス作ればいいだけである（形態素解析するクラスはすでに存在する）。他のテキスト処理と組み合わせることができるため、メリットは大きい。このようにさまざまなことを扱うことが本来的にできるため、ソースを非公開にせず、オープンソースにして公開している。

7.具体なコード例

RawlerTool は、XAML を入力し、実行できる環境を提供する(Fig.4)。XAML の作成もできるが、入力補助がある VisualStudio を使うことが望ましい。(code:1)はブログのコメントを取得する例である。Data タグにある xmlns は、使用する DLL の指定、決まり文句である。Page のところで URL を指定すると取得開始し、Tags でコメント部分だけを抽出し、そして、DataWrite で Data に対して、これは comment であるという情報を付けて書き込む。PreTree でタグを削除する前処理をしている。同様に、名前部分を取得する。ClipText は始まりと終わりを指定してそこに挟まれるテキストを取得する。

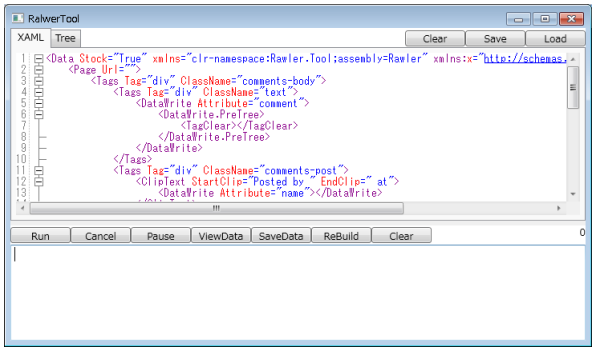

Fig. 4　RawlerTool の画面

そして、NextDataRow でそれが一つの塊であると確定し、次の塊に移る。

ページ送りに対応していて、「次のコメント」というリンクを探し、NextPage を実行すると、Page にその URL を読み込む命令をし、同じことが繰り返される。これによりすべてのコメントの取得ができることになる。

Blog の URL のところを書き変えて、これを実行すると、その記事のすべてのコメントと書いた人の名前のペアが Data に蓄積される。このように必要な記述だけで、ページの取得、繰り返し処理、データの取得が行うことができる。

```
<Data
   xmlns="clr-namespace:Rawler.Tool;assembly=Rawler"    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
   <Page Url="ブログ記事の URL">
      <Tags Tag="div" ClassName="comments-body">
         <Tags Tag="div" ClassName="text">
            <DataWrite Attribute="comment">
               <DataWrite.PreTree>
                  <TagClear></TagClear>
               </DataWrite.PreTree>
            </DataWrite>
         </Tags>
         <Tags Tag="div" ClassName="comments-post">
            <ClipText StartClip="Posted by " EndClip=" at">
               <DataWrite Attribute="name"></DataWrite>
            </ClipText>
         </Tags>
         <NextDataRow></NextDataRow>
      </Tags>
      <Links LabelFilter="次のコメント" IsSingle="True">
         <NextPage  ></NextPage>
      </Links>
   </Page>
</Data>
```

Code.1 Rawler のサンプル。ブログのコメント取得

これだけでは、毎回 Blog の URL を入れないとだめなので不便である。その時は、Page の直近の親で対象とする URL のリストつくり繰り返しを行わせればいい。繰り返す内容を直接書いてもいいし、ファイルから読み込ませるのでもいい。そのようなクラスは用意されている。

Page は親にテキストがあれば、それを URL としてアクセスし、HTML を取得する。そのため、Page→Link→Page というようにすれば、ブラウザでページ移動する感じで複数のページにアクセスして取得することが簡単に記述できる。このため、一覧ページと詳細ページがあるような構成のサイトでのデータ取得では威力を発揮する。

8.活用事例

すでにこのフレームワークを使い、１０程度のサイトからのデータ取得の実績がある。比較的容易にデータ取得ができるため、その分、分析する対象が広がる。

一例として、AKB48 の分析を挙げる。国民的アイドルとなった AKB48、ファンとのコミュニケーションのためにブログをかいている。アイドルのブログの記事そのものはアイドル相応なもので特筆することではないが、コメント欄がすごい。総選挙での上位陣は、コメント数が万単位であり、投稿時直後だけではなく継続的にコメントが書かれている。このテキストを取得するために、このフレームワークを使いデータを取得し（先ほど例示したコード）、前述の Hipparu-McS を使い分析を行った。いろいろデータを探索していく中で、それぞれのメンバーでの「ブログ、テレビ、握手」の使われ方の違いに気付き、ちょうど３つなので、それを三角グラフにプロットした。（Fig.5）

AKB について特に知らない人にはどのメンバーも同じことをしているように見えるかもしれない。しかし、このように可視化すると、ファンの反応として、AKB48 のメンバーそれぞれのメディア戦略が違うことがうかがえる。また、「僕」「私」「俺」「みんな」といった人称代名詞の使い方にも差があり、円グラフを作ってみると、それぞれで全然違うファンであることが考察できる。詳細は「アイドルブログのコメント欄から見る、「君と僕の関係」」というタイトルのブログで公開してある[4]。この記事はソーシャルブックマークサービスのはてなブックマークでは、７００近くブックマークを集める人気記事となり、ブックマークコメントでは、面白い分析だ、ブログのコメント欄に注目することが面白い、ということが多数書かれている。

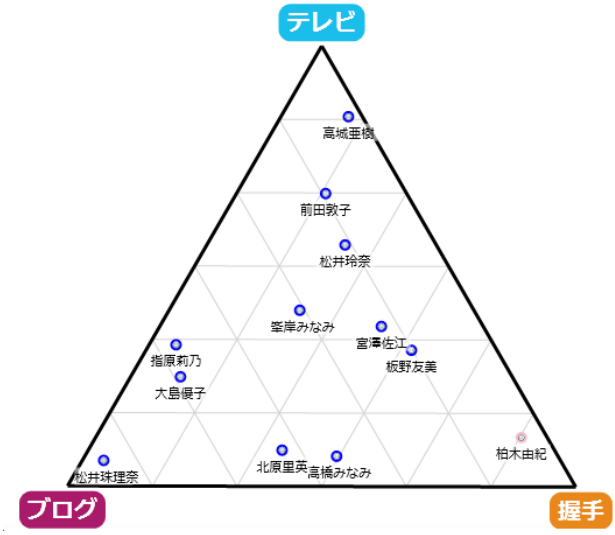

Fig. 5 AKB48 のメディア戦略の違い

このような高い評価になったのは、分析した彼女のセンスによるところが非常に大きい。このようなセンスある人に、分析できるデータをすばやく用意できるというのがすごく重要なことである。

つまり、重要なことは AKB48 の分析をしたということではなく、すばやくデータ取得し、分析できるツールを揃えることによって、分析できる世界が広がるということである。分析はトライアンドエラーの繰り返しなので、そのサイクルが速いことがいいに決まっている。

9. Rawler フレームワークの課題

著者であれば、通常のサイトのクロールを、３０分から１時間程度でこのフレームワークを使って作ることができる。しかし、現在、オープンソースで公開しているが、ドキュメントが圧倒的にすくなく、誰もが使えるというものになっていない。そのため、サンプルの数を増やして、ドキュメントを整備することが必要であろう。

また、作成のためのツールも必要である。VisualStudio での XAML の作成は完成度が非常に高いが、無償で配布されているとはいえ、多くの人に VisualStudio のインストールを必須にするのは酷であろう。結局のところ半ばむき出しのプログラミングなところがあるので、プログラマには相性が良いところがあるだろうが、初心者にも扱いやすいビジュアルプログラミングをできるようなこともできるようにする必要があるだろう。

Rawler フレームワークの価値は、データの成型の命令を柔軟に記述することができるということである。そのため、前述した、Hipparu-McS でも、現状

では一時的にテキストファイルに書き出せば分析できるが、フレームワークで記述することでデータの入力をできるようにすれば、web からの情報を直に分析するということが可能になる。データの入力のインターフェースになりうる。このようなことをできるようにしていきたい。

参考文献

[1] 伊藤貴一, 熊坂賢次, 諏訪正樹, 花房真理子「自己探求する柔らかい構造化ツール(HIPPARU-MCS)の実装と評価」, COMPUTER & EDUCATION VOL.029,2010

[2] 花房 真理子, 熊坂賢次, 伊藤貴一,「おいしさの探求ーブログのテキスト解析によるおいしさの意味世界の可視化ー」情報処理学会, 第 8 回ネットワーク生態学シンポジウム, 神奈川, 2012 年 3 月

[3] J. Hughes, Why Functional Programming Matters, In D. Turner, editor, Research Topics in Functional Programming, Addison Wesley, 1990

[4] http://web-harvest.sourceforge.net/

[5] http://d.hatena.ne.jp/haruna26/20120204/1328351411