

HTML からのテキストの自動切り出しアルゴリズムと実装

村上義継[†] 坂本比呂志^{††}
有村博紀^{††} 有川節夫^{††}

World Wide Web で収集した HTML テキストから部分的にデータを取り出すプログラムを HTML Wrapper と呼ぶ。本研究では HTML Wrapper のための新しいデータモデルを提案し、与えられた HTML から所望のテキストデータを抽出するための HTML Wrapper を自動生成する機械学習アルゴリズムを構築する。さらにこのアルゴリズムを Java によって実装し、このアルゴリズムの有効性を検証する。

Extracting Text Data from HTML Documents

YOSHITSUGU MURAKAMI,[†] HIROSHI SAKAMOTO,^{††} HIROKI ARIMURA^{††}
and SETSUO ARIKAWA^{††}

This paper introduces the new model of the HTML Wrapper for the information extraction from HTML documents and presents the learning algorithm for the HTML Wrappers in the framework of learning by examples. The expressiveness of this model is shown by experimental results.

1. はじめに

Web 上のデータからの情報抽出 (information extraction) は、半構造データに関する研究や Web マイニングにおける基本的な問題であり、その手法は Web 上の情報の探索、情報の自動抽出、Web 知識ベース、文書の自動分類、キーワード発見といった様々な分野で応用可能である。

これまで、HTML テキストからの情報抽出は Web サイトの構造に依存した Heuristics な手法を用いていたが、Kushmerick¹⁾ は計算論的手法に基づく情報抽出の枠組みを初めて提案した。これは帰納学習のひとつであり、以下に概要を説明する。

HTML テキストから必要なテキストのみを抽出するプログラムを HTML Wrapper という。抽出対象の HTML テキストをページという。ページは 1 つの長い文字列とである。ページに含まれる必要なテキストの集合を一般的な表の形にしたとき、カラム (column) に対応するものをテキスト属性と呼ぶ。またテキスト属性の集まりをタブルという。例えば住所録を記述したあるページには氏名、住所、電話番号、メールアドレスのテキスト属性があり、すべてのテキスト属性をひとつずつ含んだ組がタブルを表す。この場合 (坂本比呂志, 福岡市東区 ..., 092-642..., hiroshi@...) や (有村博紀, 福岡市西区 ..., 092-642...,

arim@...) がタブルである。また, hiroshi@... をこの 1 番目のタブルの 4 番目のテキスト属性という。そして、ページ中のテキスト属性の位置を指定したものをラベルとして定義する。

Kushmerick はいくつかの HTML Wrapper のクラスを定義しているが、その中で最も基本的な LR Wrapper を説明する。その他の Wrapper については論文¹⁾ を参照。LR Wrapper の一般形は $\langle (l_1, r_1), \dots, (l_K, r_K) \rangle$ であり、各 (l_i, r_i) は文字列である。LR Wrapper を用いたテキスト抽出アルゴリズムは、ページから最初の l_1 と r_1 の出現を探してその間の文字列を 1 番目のテキスト属性として抽出する。ページ中のテキスト属性の数を K とすると、同様に K 番目の属性までテキスト属性を抽出した後、それらを 1 つのタブルとする。次の l_1 が存在しなくなるまで同様に行う。

LR Wrapper を学習するアルゴリズムは、 i 番目のテキスト属性の左側に共通する最長接尾語を l_i 、右側に共通する最長接頭語を r_i として求めていく。

以上のような方法だと l_i や r_i が "" などのように極端に短くなった場合、ページ中のテキスト属性以外の場所も抽出してしまう可能性が高くなる。Kushmerick の他の HTML Wrapper クラスについても同様の問題が生じる。これはページを平面的なテキストとして扱う限り避けられない問題である。

そこで、本研究ではページを木構造に展開し、その上でテキスト抽出を行うアルゴリズムと、その抽出ルールである Tree-Wrapper の学習アルゴリズムを提案する。そしてアルゴリズムを、Java と、XML や HTML のパーサ

[†] 九州大学大学院システム情報科学研究科情報理工学専攻

^{††} 九州大学大学院システム情報科学研究科情報理工学部門
Department of Informatics, Graduate School of Information Science and Electrical Engineering, Kyushu University

である DOM²⁾ (Document Object Model) によって実装し、実際のページについて有効性を検証する。

2. テキスト抽出のためのデータモデル

本節では、HTML から木構造データを構築するためのデータモデルを定式化する。

2.1 HTML 木とノードの定義

ページ P はノードにラベルを持つ根付き順序木 P_ℓ で表現することができる。本節ではこの P_ℓ を構成するための厳密な定義を与える。ただし、ページ P からコメントはあらかじめ取り除かれていると仮定する。

ある木 T がノード n を根とし、 n が $n_1, \dots, n_k$ の子供をこの順番で持つとき T は式 $n(n_1, \dots, n_k)$ と等価である。 P_ℓ のノードの集合は自然数のある有限集合である。すべてのノードは、要素ノードまたはテキストノードのいずれかに分類される。またノード n に付けられているノードラベル $NL(n)$ は、 $NL(n) = \langle N(n), V(n), HAS(n) \rangle$ で定義される。 $N(n)$ はノード名 (Name) で $V(n)$ はノード値 (Value) を表す。 n が要素ノード場合はノード名はタグ名であり、ノード値は空である。 n がテキストノードの場合はノード名は特別な名前 $\text{\#TEXT}$ であり、ノード値はそのテキストである。さらに、 $HAS(n) = \{HA_1, \dots, HA_\ell\}$ は要素ノード n に対して定義される HTML 属性の集合であり、 $HA_i = \langle a_i, v_i \rangle$ ($i = 1, \dots, \ell$) の形をしている。ここで a_i を HTML 属性名、対応する v_i を a_i の HTML 属性値という。

定義 1 任意のページ P に対して、 P_ℓ は以下で定義される順序木である。

- (1) ページ P のすべての空タグ $\langle A \rangle$ は P_ℓ の子供を持たないある要素ノード n に対応する。ただし $NL(n) = \langle N(n), V(n), HAS(n) \rangle$, $N(n) = A$, $V(n) = \epsilon$, $HAS(n) = \emptyset$ である。
- (2) ページ P の任意の文字列 $t_1 \cdot w \cdot t_2$ に対し、 w がタグを含まない文字列、 t_1 と t_2 がタグであるとき、 w は P_ℓ のテキストノード n に対応する。ただし $NL(n) = \langle N(n), V(n), HAS(n) \rangle$, $N(n) = \text{\#TEXT}$, $V(n) = w$, $HAS(n) = \emptyset$ である。
- (3) ページ P に含まれる開始タグと終了タグで囲まれた文字列 $\langle A \ a_1 = v_1 \dots a_\ell = v_\ell \rangle w \langle /A \rangle$ は P_ℓ の部分木 $t = n(n_1, \dots, n_k)$ に対応する。ただし $NL(n) = \langle N(n), V(n), HAS(n) \rangle$, $N(n) = A$, $V(n) = \epsilon$, $HAS(n) = \{\langle a_1, v_1 \rangle, \dots, \langle a_\ell, v_\ell \rangle\}$ であり、 $n_1, \dots, n_k$ はそれぞれ、文字列 w に対して (1), (2), (3) で帰納的に定義される木 $t_1, \dots, t_k$ の根である。

以下の議論では、 P_ℓ をページと呼ぶことにする。

2.2 木のノードを取り出す関数

ここでは上で構成したページ P_ℓ のノードに対していくつかの関数を簡単に説明する。これらの関数は指定された引数に対して定義され、説明されている値を返す。また

その値が存在しないときは null が返される。

- Parent(n): ノード n の親。
- ChildNodes(n): n の子供の列。
- Name(n): n のノード名 $N(n)$ 。
- Value(n): n を頂点とする木のすべてのテキストノード $n_i, n_j, \dots, n_k$ に対してノード値の連結 $V(n_i) \cdot V(n_j) \dots V(n_k)$ 。
- Pos(n): n のすべての兄弟を $n_1, \dots, n_i, \dots, n_k$ ($n_i = n$) とするとき、 $Name(n_j) = Name(n)$ ($j \leq i$) であるような n_j の個数。

2.3 HTML 属性を取り出す関数

ノードと同様に P_ℓ の HTML 属性を操作するためのいくつかの関数を定義する。

- HTMLAttrSet(n): ノード n の HTML 属性集合 $HAS(n)$ 。
- HTMLAttrName(n, i): $HAS(n)$ の添え字 i の要素 $\langle a_i, v_i \rangle$ の HTML 属性名 a_i 。
- HTMLAttrValue(n, i): $HAS(n)$ の添え字 i の要素 $\langle a_i, v_i \rangle$ の HTML 属性値 v_i 。
- HTMLAttrValueByName($n, name$): $HAS(n)$ のある要素 $\langle name, value \rangle$ の HTML 属性値 $value$ 。

最後にノード $n_1, \dots, n_k$ の HTML 属性集合の集合 $S = \{HAS(n_1), \dots, HAS(n_k)\}$ に対して、共通 HTML 属性集合 $CHAS(S)$ と、それに対する関数を定義する。

定義 2 集合 $S = \{HAS(n_1), \dots, HAS(n_k)\}$ および $HAS(n_i) = \{\langle a_{i1}, v_{i1} \rangle, \dots, \langle a_{i\ell}, v_{i\ell} \rangle\}$ ($i = 1, \dots, k$) に対して、共通 HTML 属性集合 $CHAS(S)$ は以下で定義される HTML 属性集合である。

すべての $i = 1, \dots, k$ に対して、 $a_{n_i} = name$ なる HTML 属性名 $name$ が存在して、

- (1) すべての $i = 1, \dots, k$ に対して $v_{n_i} = value$ なる HTML 属性値 $value$ が存在する。
 $\equiv_{def} \langle name, value \rangle \in CHAS(S)$ 。
- (2) それ以外、 $\equiv_{def} \langle name, * \rangle \in CHAS(S)$ 。

ただし $*$ はすべての値にマッチする特別な記号 (ワイルドカード) である。

最後に共通 HTML 属性集合を返す関数を次のように定義する。

- CommonAttrSet ($HAS(n_1), \dots, HAS(n_k)$): HTML 属性集合 $HAS(n_i)$ ($i = 1, \dots, k$) の共通 HTML 属性集合。

2.4 テキスト属性、タプル、ラベル

HTML Wrapper がページ P_ℓ から抽出する個々のテキストは、あるテキストノード n のノード値 $V(n)$ である。このノード n をテキスト属性 (text attribute) と呼ぶ。一般にはページ P_ℓ では、あるタプルの k 番目のテキスト属性の数が 0、あるいは複数の場合がある。例えば住所録データの k 番目のテキスト属性を電話番号とすると、ある人が電話番号を 2 つ持っている場合や未記入の場合が想定される。このような場合に対応できるように、

テキスト属性をノードの集合 $ta \subseteq \{0, \dots, m\}$ に拡張する。そしてテキスト属性の列 $t = \langle ta_1, \dots, ta_k \rangle$ をタプルといい、さらにタプルの集合 $L = \{t_1, \dots, t_n\}$ をラベルと呼ぶ。

3. Tree-Wrapper

未知のページからテキスト属性を抽出するには、そのテキスト属性が持つ一般的なルールを発見しなければならない。本節ではそのようなルールを Tree-Wrapper として木のパス (路) で表現する手法を提案する。そして、2つのアルゴリズムを与える。1つは Tree-Wrapper を用いてページからラベルを抽出するアルゴリズム **execT** で、もう1つはこの Tree-Wrapper を与えられたページとそのラベルから推論する学習アルゴリズム **learnT** である。

3.1 Tree-wrapper

定義3 $ENL = \langle N, Pos, HAS \rangle$ を抽出ノードラベル (extraction node label) という。ただし N はノード名、 Pos は自然数または $*$ 、 HAS は HTML 属性の有限集合である。さらに抽出ノードラベルの列 $EP = \langle ENL_1, \dots, ENL_\ell \rangle$ を抽出パス (extraction path) という。

個々の抽出ノードラベル ENL はそれらを含む抽出パス EP が、与えられた P_t のあるパスにマッチするかどうかを判定するために用いられる。その判定の意味論は次の関数 **isMatchENL** が返す値によって決定される。

```

boolean isMatchENL( $n, ENL$ )
/*input: ノード  $n$ ,  $ENL = \langle N, Pos, HAS \rangle$  */
/*output: true or false */
if ( $N == Name(n)$  && ( $Pos == Pos(n) \parallel Pos == *$ ) &&
    isMatchHAS( $n, HAS$ )) == true;
else return false;

boolean isMatchHAS( $n, HAS$ )
/*input: ノード  $n$ ,  $HAS = \langle HA_1, \dots, HA_m, \dots, HA_M \rangle$  */
/* $HA_m = \langle a_m, v_m \rangle$  */
/*output: true or false */
for ( $m=1; m \leq M; m++$ ) {
    if (HTMLAttrValueByName( $n, a_m$ )  $\neq v_m$  &&  $v_m \neq *$ )
        return false;
} return true;

```

定義4 ENL を抽出ノードラベルとし、 n をページ P_t のノードとするとき、関数 **isMatchENL** が真であるならば ENL は n にマッチするという。さらに抽出パスを $EP = \langle ENL_1, \dots, ENL_\ell \rangle$ とし、ページ P_t のパスを $p = \langle n_1, \dots, n_\ell \rangle$ とすると、すべての $i = 1, \dots, \ell$ に対して ENL_i が n_i にマッチするとき EP は p にマッチするという。

3.2 Tree-Wrapper を用いた抽出

抽出アルゴリズムの目標は、このような抽出パス EP_t

にマッチするパスを探してマッチした最後のノードを i 番目のテキスト属性として取り出すことである。逆に学習アルゴリズムの目標は、 i 番目のテキスト属性が複数指定されたときにそれらを全て抽出することができる抽出パス EP_i を発見することである。

定義5 Tree-Wrapper とは、 $W = \langle EP_1, \dots, EP_K \rangle$ である。ここで $EP_i = \langle ENL_1^i, \dots, ENL_{\ell_i}^i \rangle$ は抽出パスであり、 ENL_j^i は抽出ノードである。

次に Tree-Wrapper W とページ P_t からラベルの抽出を実行するアルゴリズム **execT**(P_t, W) を説明する。

Algorithm execT(P_t, W)

```

/*input:  $W = \langle EP_1, \dots, EP_K \rangle$ , ページ  $P_t$  */
/*output:  $K$  個のテキスト属性を持つタプル  $t_i$  の集合
 $L_t = \{t_1, \dots, t_m\}$  ( $P_t$  のラベル) */
(1)  $EP_i = \langle ENL_1^i, \dots, ENL_{\ell_i}^i \rangle$  ( $i=1, \dots, K$ ) にマッチする  $P_t$  の任意のパス  $p$  の最後のノード  $n$  に対して、組  $\langle i, n \rangle$  を集合  $Att$  に加える。
/* $Att$  はテキスト属性の候補集合で、これを以下で各タプルに分配する。*/
(2) 集合  $Att$  のすべての要素をノード番号  $n$  に関して昇順にソートする。そのリストを  $LIST$  とし、 $j=1$  とする。
(3)  $LIST$  の長さが 0 であるならば停止する。そうでないならば  $i_\ell$  が減少しない  $LIST$  の最長接頭辞列  $list = \langle i_1, n_1 \rangle, \dots, \langle i_\ell, n_\ell \rangle$  を見つけて、すべての  $i = 1, \dots, K$  に対して  $ta_i = \{n \mid \langle i, n \rangle \in list\}$  を計算する。ただし  $\langle i, n \rangle$  が存在しないときは  $ta_i = \emptyset$  とする。
(4)  $t_j = \langle ta_1, \dots, ta_K \rangle$ ,  $j=j+1$  とし、 $LIST$  から  $list$  を取り除いて (3) へ戻る。

```

3.3 Tree-Wrapper の学習

Tree-Wrapper $W = \langle EP_1, \dots, EP_K \rangle$ の学習アルゴリズム **learnT**(E) を定義する。ただし E はページとラベルの組 $\langle P_{t_n}, L_n \rangle$ の有限集合である。アルゴリズム中の関数 **learnExPath** によって、任意のテキスト属性 $i = 1, \dots, K$ を抽出する抽出パス $EP_1, \dots, EP_K$ を発見する。アルゴリズム **learnT** は訓練例 $\langle P_{t_n}, L_n \rangle$ が与えられるたびに、任意の $k = 1, \dots, K$ に対して P_{t_n} から k 番目の属性を抽出するための抽出パス EP_k^n を計算する。そしてその EP_k^n をこれまでの計算結果である EP_k と合成する。合成 $EP_k \cdot EP_k^n$ の定義は定義6で与える。

また関数 **getPath**(n) は与えられたノード n からその木の根までのパスに現れる任意のノード n_i に対して、抽出ノードラベル $ENL_i = \langle N_i, Pos_i, HAS_i \rangle$ を構成し、抽出パス $EP = \langle ENL_\ell, \dots, ENL_1 \rangle$ を返す。ただし ENL_i は根から順に並んでおり、 n に対する抽出ノードラベルは ENL_1 である。

定義6 ENL_1, ENL_2 を抽出ノードラベルとすると、合成 $ENL_1 \cdot ENL_2$ はつぎの条件を満たす抽出ノードラベル $ENL = \langle N, Pos, HAS \rangle$ である。

- (1) $N_1 = N_2$ ならば $N = N_1$. それ以外は ENL は無定義.
- (2) $Pos_1 = Pos_2$ ならば $Pos = Pos_1$. それ以外のときは $Pos = *$.
- (3) $HAS = \text{CommonAttSet}(HAS_1, HAS_2)$.

また異なる抽出パス $EP_1 = \langle ENL_1^1, \dots, ENL_1^1 \rangle$, $EP_2 = \langle ENL_2^1, \dots, ENL_2^1 \rangle$ の合成 $EP_1 \cdot EP_2$ は, すべての合成 $ENL_i^1 \cdot ENL_i^2$ が定義可能な最大の列 $\langle ENL_i^1 \cdot ENL_i^2, \dots, ENL_1^1 \cdot ENL_1^2 \rangle$ ($i = 1, \dots, \ell$) である. ただし $\ell \leq \min\{n, m\}$.

```

learnT(E)
/*input: E={...,<Ptn,Ln>,...}*/
/*output: すべての<Ptn,Ln>に対して*/
/*execT(Ptn,W)=LnとなるTree-Wrapper W*/
Ln= $\{t_i \mid 1 \leq i \leq M_n\}$ ; /*ラベル*/
ti= $\langle ta_1^i, \dots, ta_K^i \rangle$ ; /*タプル*/
NODE(n,k)= $\cup_{1 \leq i \leq M_n} ta_k^i$ ; /*属性kのすべてのノード集合*/
n=1;
do while( $\langle P_{t_{n+1}}, L_{n+1} \rangle$ ){
  for(k=1; k ≤ K; k++){
    if(NODE(n,k) && EPkn=leanExPath(NODE(n,k)
      && EPk=ε EPk=EPkn;
    if(NODE(n,k) && EPkn=leanExPath(NODE(n,k)
      && EPk != ε EPk=EPk · EPkn;
    else ;
  } n=n+1;
} return W= $\langle EP_1, \dots, EP_K \rangle$ ;

leanExPath(NODE(n,k))
/* input: NODE(n,k)= $\{n_1, \dots, n_m\}$  */
/* output: 抽出パス EP */
if(m==1) return EP=getPath(n1);
else j=1;
for(i=1; i ≤ m; i++) currenti=ni;
while( $\forall i \text{ Name}(current_i) == \text{Name}(current_1)$ ){
  ENLj= $\langle N_j, Pos_j, HAS_j \rangle$ ;
  Nj=Name(current1);
  if( $\forall i \{Pos(current_i) == Pos(current_1)\}$ )
    Posj=Pos(current1);
  else Posj=*;
  HASj=CHAS(S);
  S= $\{HTMLAttSet(current_i) \mid i = 1, \dots, m\}$ ;
  for(i=1; i ≤ m; i++) currenti=Parent(currenti);
  j=j+1;
}
if(j==1) return false;
else return EPk= $\langle ENL_{j-1}, \dots, ENL_1 \rangle$ ;

```

4. 実験

本論文で構築した,Tree-Wrapper を用いた抽出アルゴリズムと Tree-Wrapper の学習アルゴリズムを実装し,比較的少量の HTML テキストに対して以下の実験を行った.

インターネットで公開されている論文データベースである citeseers (<http://citeseer.nj.nec.com>) から 1300 個の論文ページを収集し,それぞれ $P_{t_1}, \dots, P_{t_{1300}}$ とした. このうち $P_{t_1}, \dots, P_{t_{10}}$ に対し, 3 つのテキスト属

性 (論文名, 著者名, アブストラクト) の位置を指定したラベル $L_1, \dots, L_{10}$ を作成した. 次に, $\dots, \langle P_{t_n}, L_n \rangle, \dots$ ($1 \leq n \leq 10$) を訓練例として学習プログラムに与え, 図 1 のような Tree-Wrapper $W = \langle EP_1, EP_2, EP_3 \rangle$ を得た.

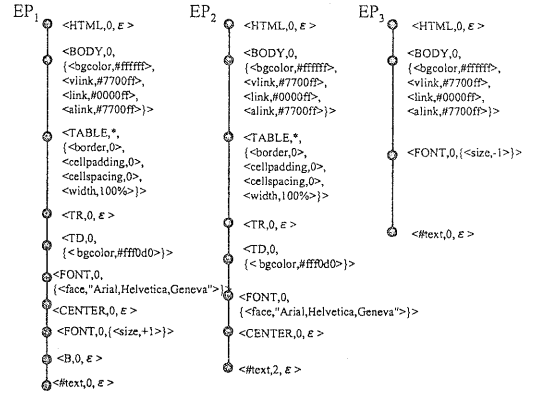


図 1 学習した Tree-Wrapper

そして, $L_n = \text{execT}(P_{t_n}, W)$ ($11 \leq n \leq 1300$) として $P_{t_{11}}, \dots, P_{t_{1300}}$ からラベル $L_{11}, \dots, L_{1300}$ を抽出した. それらのラベル L_n を検証したところ, P_{t_n} 中の 3 つのテキスト属性の位置を正しく指定していなかったラベルは特殊な場合の 3 つだけであった.

以上のことから, これらのページに関して, 学習した Tree-Wrapper は十分有効であることがわかった.

5. おわりに

本論文では Web 上に大量に存在する HTML テキストから部分的なテキストを抽出するプログラムである Tree-Wrapper を提案した. また, このアルゴリズムを実装し, 実際の Web 上の HTML テキストから部分テキストの抽出の実験を行った. その結果, siteceers では, 十分実用的であることがわかった.

今後の課題は, より多くのサイトについて実験を行い, Tree-Wrapper の一般性を検証することである. その際に以下のようなモデルの拡張を考えている. 第一に, 抽出パスとパスの最後のノードの HTML 属性名の組によって, HTML 属性値も抽出できるモデルに拡張したい. また, 現在のモデルで抽出したテキストノードから, さらに部分テキストを抽出できるようにしたい. 例えば, そのノードがテキスト “日付 (曜日)” であるときに “日付” と “曜日” を別々に抽出することを目指す.

参考文献

- 1) N. Kushmerick: Wrapper induction: efficiency and expressiveness, Artificial Intelligence, Vol.118, pp.15–68, 2000.
- 2) World Wide Web Concorsiam, <http://www.w3.org/DOM>