Inferring Grammars from Semi-structured Documents

Project Report

Author:
Talya Kalimi

Supervisor:
Dr. Roman Manevich

December 7, 2018

Ben-Gurion University

Department of Computer Science

1 Introduction

Project Goal:
Develop tool to automatically learn grammar from a set of semi-structured documents.

Motivation: semi-structured documents:
Extremely prevalent: XML, JSON, CSV, log files...

- The good: provide a meta-level grammar
- The bad: often not accompanied by grammar for each use case

Problems:

- Hard to spot syntax errors
- Hard to develop generic tools

High-level ideas
1. Tokenize the input: identify different types of substrings (tokens)
   - Punctuation
   - Fields
   - Data
2. Identify sequences of tokens that induce parenthetical behavior
3. (incomplete) Build grammar from derivation tree for each input

The following sections describes the algorithms main idea, the uses and gives some examples.
You can find the source code here
All algorithms supporting accept hints from the user to guide the learner.
please read README.md file.

2 Inferring punctuation tokens

I used haid/tail breaks algorithm which get a set of examples and partition them into clusters - an unsupervised learning technique.
Time complexity: O(n^2), bounded by computing all consecutive substrings

1. $S \leftarrow \emptyset$
2. *for every substring* m :
   if $|m| < k$
      if$\{(m, freq)\} \in S$
         $S \leftarrow S \cup \{(m, freq + 1)\}$
      else $S \leftarrow S \cup \{(m, 1)\}$
3. *sort* S *by ascending order of* $freq$
4. $H \leftarrow \emptyset$
5. $H \leftarrow$ **head/tail Breaks**(S, H)

Recursive function head/tail Breaks(S, H)
//the head for data values greater the mean
//the tail for data values less the mean

if $|H| < |S| * 0.4$
   Break S (*around mean*) *as a whole into the* **head** *and the* **tail**
   $H \leftarrow H \cup$ *head*
   head/tail Breaks $(tail, H)$
else return H

H is the group of punctuation tokens.

3 Inferring fields tokens

Fields appears many times in semi-structured documents.
Thus, I used algorithm to find Longest Repeated Substring that occurs in a string, using suffix tree
Time complexity: O(nlogn), bounded by suffix array construction

1. *let T be the tokenized text*
2. *remove all punctuations from T*
3. *build SuffixArray SA (from T) using Manber & Myers algorithm* [1]
3. *use Kasai algorithm* [2] *to build LCP (longest common prefix) array*
4. $F \leftarrow \emptyset$
5. $LRS \leftarrow FindLrs(SA)$
6. *while* $LRS \neq \emptyset$
   $F \leftarrow F \cup \{LRS\}$
   remove $\{LRS\}$ *from T*
   build new SuffixArray SA (from T) using Manber & Myers algorithm
   $LRS \leftarrow$ **FindLrs**(SA)
7. *return F*

Function FindLrs (SA)
//Finds the LRS(s) (Longest Repeated Substring) that occurs in a string
//return an ordered set of longest repeated substrings

1. $maxLen \leftarrow 0$
2. $LRS \leftarrow \emptyset$
3. *for* $i \leftarrow 1$ *to* $|LCP|$
   if $LCP[i] > maxLen$
      $LRS \leftarrow \emptyset$
   else
      $s \leftarrow$ *string represent by* $SA[i]$
      $maxLen \leftarrow LCP[i]$
      $LRS = LRS \cup \{s\}$
4. *return LRS*

F is the group of Fields tokens

4 Inferring data tokens

After inferring punctuation and fields, the remain tokens are categorized as data
Time complexity: O(n)

5 Inferring parentheses

Bottom-up algorithm to find parenthesis-type tokens, using heuristic based on distance between them.
Every Parenthesis can assembled from combination of k tokens (some small k).
Time complexity: O(n^2), bounded by computing all possible pairs and check if are parenthesis-type

1. $S \leftarrow \emptyset$ *//group of single token*
2. $P \leftarrow \emptyset$ *//group of pairs of tokens*
3. $T \leftarrow \emptyset$ *//group of combination of tree token*
4. $B \leftarrow \emptyset$
5. *for every token in tokenized text*
   if token $\in$ *Fields or token* $\in$ *Punctuations*
   $S \leftarrow S \cup \{token\}$
6. *for* $(i, j) : i < j$
   $B \leftarrow$ **checkBalancePair**$(S[i], S[j], B)$
   $P \leftarrow P \cup \{S[i] + S[j]\}$
7. *for* $(i, j) : i < j$
   $B \leftarrow$ **checkBalancePair**$(P[i], P[j], B)$
8. *for* $(i, j) : i < j$
   $B \leftarrow$ **checkBalancePair**$(S[i], P[j], B)$
   $T \leftarrow T \cup \{S[i] + P[j]\}$
9. *for* $(i, j) : i < j$
   $B \leftarrow$ **checkBalancePair**$(T[i], T[j], B)$
10. $B \leftarrow$ **FilterPairs**(B)

Function checkBalancePair(right, left,B)
using stack check if right and left are balanced parentheses
1. *Stack S*
2. *for every token in tokenized text*
   if token = right
   S.push(token)
   else if token = left
   S.pop()
3. *if S.isEmpty()*
   $B \leftarrow B \cup \{(right, left)\}$
4. *return B*

FilterPairs(B)
//position(x) is x position in tokenized text
$k \leftarrow \epsilon$ *//choose some small number*
$\forall(right, left) \in B$ *do*
$if(position(left) - position(right)) > k$
$B \leftarrow B \setminus (right, left)$

B is the group of pairs of parenthetical behavior tokens

6 Grammar learning

(incomplete)
A Grammar-based approach towards unifying hierarchical data model.
Incremental learning.
I implemented two major rules [3]:

MergeNonTerminals(N1,N2)

1.$if\ N1 \rightarrow \alpha A \beta\ and\ N1 \rightarrow \alpha B \beta$
$A = B$

2. $if\ N1 \rightarrow \alpha\ and\ N2 \rightarrow \alpha$
$N1 = N2$

7 XML Example

```
<CATALOG>
  <CD>
        <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  <CD>
    <TITLE>Greatest Hits</TITLE>
    <ARTIST>Dolly Parton</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>RCA</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1982</YEAR>
  </CD>
  <CD>
    <TITLE>Still got the blues</TITLE>
    <ARTIST>Gary Moore</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Virgin records</COMPANY>
    <PRICE>10.20</PRICE>
    <YEAR>1990</YEAR>
  </CD>
  <CD>
    <TITLE>Eros</TITLE>
    <ARTIST>Eros Ramazzotti</ARTIST>
    <COUNTRY>EU</COUNTRY>
    <COMPANY>BMG</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1997</YEAR>
  </CD>
  <CD>
    <TITLE>One night only</TITLE>
```

```
  <ARTIST>Bee Gees</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Polydor</COMPANY>
  <PRICE>10.90</PRICE>
  <YEAR>1998</YEAR>
</CD>
<CD>
  <TITLE>Sylvias Mother</TITLE>
  <ARTIST>Dr.Hook</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>CBS</COMPANY>
  <PRICE>8.10</PRICE>
  <YEAR>1973</YEAR>
</CD>
<CD>
  <TITLE>Maggie May</TITLE>
  <ARTIST>Rod Stewart</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>Pickwick</COMPANY>
  <PRICE>8.50</PRICE>
  <YEAR>1990</YEAR>
</CD>
<CD>
  <TITLE>Romanza</TITLE>
  <ARTIST>Andrea Bocelli</ARTIST>
  <COUNTRY>EU</COUNTRY>
  <COMPANY>Polydor</COMPANY>
  <PRICE>10.80</PRICE>
  <YEAR>1996</YEAR>
</CD>
<CD>
  <TITLE>When a man loves a woman</TITLE>
  <ARTIST>Percy Sledge</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <COMPANY>Atlantic</COMPANY>
  <PRICE>8.70</PRICE>
  <YEAR>1987</YEAR>
</CD>
<CD>
  <TITLE>Black angel</TITLE>
  <ARTIST>Savage Rose</ARTIST>
  <COUNTRY>EU</COUNTRY>
  <COMPANY>Mega</COMPANY>
  <PRICE>10.90</PRICE>
  <YEAR>1995</YEAR>
</CD>
```

```
  <CD>
    <TITLE>1999 Grammy Nominees</TITLE>
    <ARTIST>Many</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Grammy</COMPANY>
    <PRICE>10.20</PRICE>
    <YEAR>1999</YEAR>
  </CD>
  <CD>
    <TITLE>For the good times</TITLE>
    <ARTIST>Kenny Rogers</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Mucik Master</COMPANY>
    <PRICE>8.70</PRICE>
    <YEAR>1995</YEAR>
  </CD>
  <CD>
    <TITLE>Big Willie style</TITLE>
    <ARTIST>Will Smith</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1997</YEAR>
  </CD>
  <CD>
    <TITLE>Tupelo Honey</TITLE>
    <ARTIST>Van Morrison</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Polydor</COMPANY>
    <PRICE>8.20</PRICE>
    <YEAR>1971</YEAR>
  </CD>
</CATALOG>
```

7.1 Inferring punctuation tokens

At first, S is all substrings of length $\leq$ k (see Figure 1)
After using the Inferring-punctuation-tokens algorithm (see Figure 2),
we get the punctuation:

```
< > <\ \
```

7.2 Inferring fields tokens

After using the Inferring-fields-tokens algorithm,
we get the fields:

COMPANY, ARTIST, PRICE, TITLE, YEAR, COUNTRY, CD

7.3 Inferring parentheses

After using the Inferring-parentheses-tokens algorithm, including filters, we get the pairs:

```
(<PRICE>, </PRICE>)
(PRICE>, <\PRICE)
(<TITLE>, </TITLE>)
(TITLE>, <\TITLE)
(<YEAR>, </YEAR>)
(YEAR>, <\YEAR)
(<COMPANY>, </COMPANY>)
(COMPANY>, <\COMPANY)
(<COUNTRY>, </COUNTRY>)
(COUNTRY>, <\COUNTRY)
(<CD>, </CD>)
```

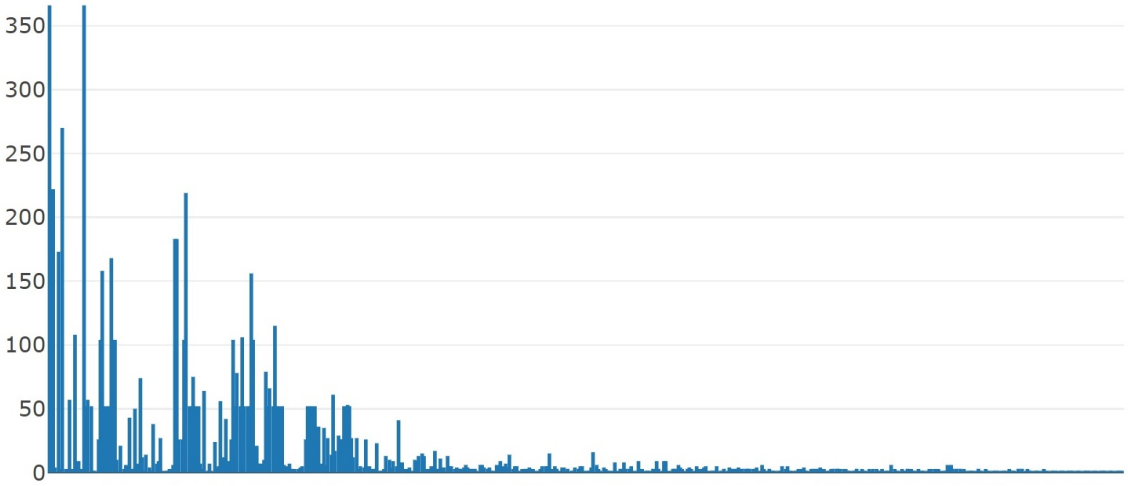

Figure 1: Histogram of all possible punctuation at first iteration

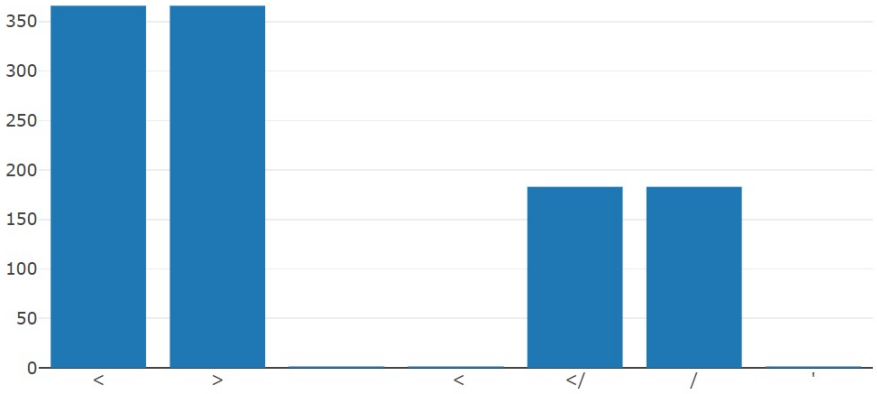

Figure 2: Histogram of all possible punctuation at last iteration

8 JSON Example


```
{
  "clients": [
    {
      "id": "59761c23b30d971669fb42ff",
      "isActive": true,
      "age": 36,
      "name": "Dunlap Hubbard",
      "gender": "some",
      "company": "CEDWARD",
      "email": "dunlaphubbard@cedward.com",
      "phone": "+1 (890) 543-2508",
      "address": "169 Rutledge Street Konterra Northern Mariana Islands 8551"
    },
    {
      "id": "59761c233d8d0f92a6b0570d",
      "isActive": true,
      "age": 24,
      "name": "Kirsten Sellers",
      "gender": "uni",
      "company": "EMERGENT",
      "email": "kirstensellers@emergent.com",
      "phone": "+1 (831) 564-2190",
      "address": "886 Gallatin Place Fannett Arkansas 4656"
    },
    {
      "id": "59761c23fcb6254b1a06dad5",
      "isActive": true,
      "age": 30,
      "name": "Acosta Robbins",
      "gender": "male",
      "company": "ORGANICA",
      "email": "acostarobbins@organica.com",
      "phone": "+1 (882) 441-3367",
      "address": "697 Linden Boulevard Sattley, Idaho 1035"
    },
    {
      "id": "63598c23b30d971669xz42ws",
      "isActive": false,
      "age": 50,
      "name": "Talya Kalimi",
      "gender": "female",
      "company": "GOOGLE",
      "email": "tkalimi@cedward.com",
```

```
    "phone": "+1 (890) 543-2508",
    "address": "169 Rutledge Street Konterra Northern Mariana Islands 8551"
  },
  {
    "id": "59761b123k8d0f78a6b0570d",
    "isActive": false,
    "age": 18,
    "name": "Dana Halperin",
    "gender": "uni",
    "company": "FACEBOOK",
    "email": "danahalp@emergent.com",
    "phone": "+1 (831) 564-2190",
    "address": "886 Gallatin Place Fannett Arkansas 4656"
  },
  {
    "id": "59761c23fcb6talya6375ks5",
    "isActive": true,
    "age": 47,
    "name": "Jack Tompson",
    "gender": "male",
    "company": "MICROSOFT",
    "email": "tompson47@organica.com",
    "phone": "+1 (882) 441-3367",
    "address": "697 Linden Boulevard Sattley, Idaho 1035"
  }
 ]
}
```


8.1 Inferring punctuation tokens

At first, S is all substrings of length $\leq$ k (see Figure 3)
After using the Inferring-punctuation-tokens algorithm (see Figure 4),we get the punctuation:

" : , } { -

8.2 Inferring fields tokens

After using the Inferring-fields-tokens algorithm,
we get the fields:
address, phone, age, email, name, isActive, true, id

8.3 Inferring parentheses

After using the Inferring-parentheses-tokens algorithm, including filters,
we get the pairs: (" ,")

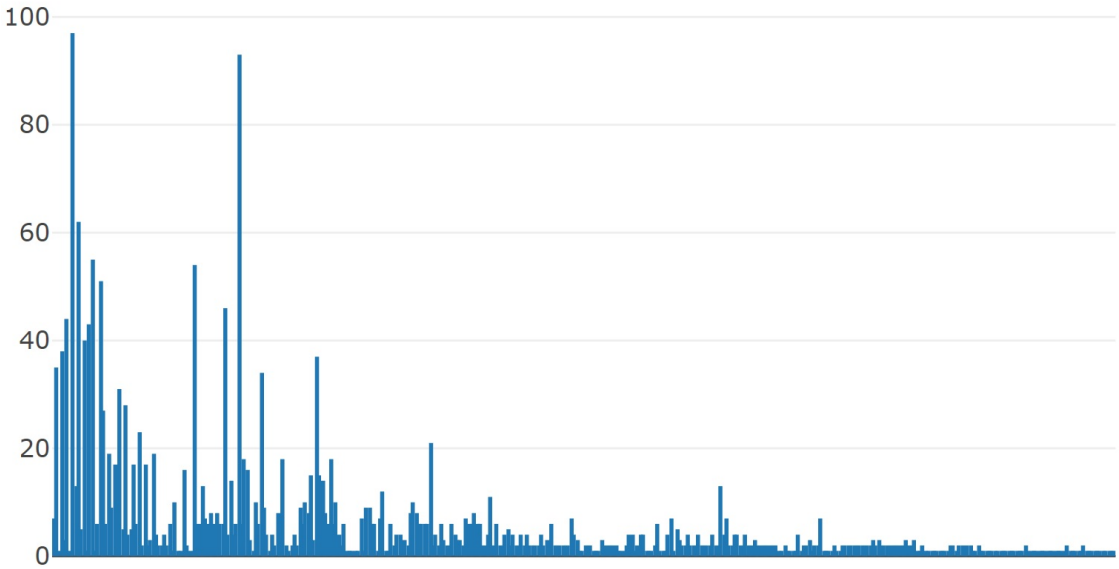

Figure 3: Histogram of all possible punctuation at first iteration

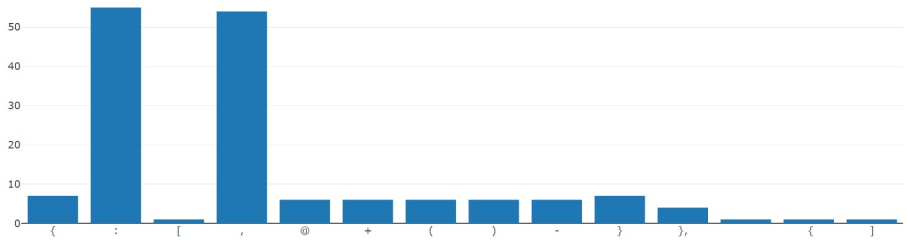

Figure 4: Histogram of all possible punctuation at last iteration

9 Bibliography

1. Manber, Udi; Myers, Gene (1993). "Suffix arrays: a new method for on-line string searches". SIAM Journal on Computing. 22: 935–948

2. Kasai, T.; Lee, G.; Arimura, H.; Arikawa, S.; Park, K. (2001). Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications. Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching. Lecture Notes in Computer Science. 2089.pp. 181–192

3. Learning k-Reversible Contex-Free Grammars from Positive Structural Examples