Wprowadznie do xml'a w R

Zygmunt Zawadzki z.zawadzki@erkakrakow.pl

W razie niejasności (coś nie zostało wytłumaczone wyastarczająco jasno, lub brak jest wystarczającej liczby odniesień do materiałów zewnętrznych, albo pominąłem istotny wątek, ewentualnie masz pomysł na jakiś ciekawy przykład) proszę pisać na podany wyżej adres e-mail, albo wykorzystać GitHuba - https://github.com/eRkaKrakow/Tutoriale/issues. Postaram się w miarę możliwości wprowadzić potrzebne uzupełnienia (jeżeli ktoś ma ochotę może również coś dopisać, to współautorzy są mile widziani - sława czeka!). Przepraszam za literówki:)

Wprowadzenie

XML to rodzaj języka znaczników, bardzo podobny do html (w zasadzie jeżeli ktoś widział na oczy html, to z xmlem poczuje się jak w domu).

Jedną z głównych zalet xml'a jest to, że jest dosyć łatwy do czytania zarówno dla człowieka, jak i dla maszyny. Poniżej znajduje się prosty przykład xml'a opisujący model regresji liniowej.

```
<?xml version="1.0" encoding="UTF-8"?>
<model>
  <!-- Opis xml modelu regresji liniowej -->
  <coefficients>
    <coeff name="Intercept" value="20.53" sd="11.01" intercept="true"/>
    <coeff name="price" value="30.1" sd="2.06" signif="true"/>
  </coefficients>
  <sd value="5.43"/>
  <rsqr>0.45</rsqr>
</model>
```

W zasadzie opierając się na samej informacji, że jest to regresja liniowa (oczywiście trzeba znać troszkę statystyki, żeby wiedzieć, że co to jest ta regresja:)), a nie znając budowy pliku xml, można łatwo domyślić się, że mamy do czynienia z modelem postaci:

$$y = 20.53 + 30.1 \cdot price, \tag{1}$$

a R^2 tego modelu wynosi 0.45.

Podstawowe określenia w xml'u.

Po krótkiej prezentacji xml'a, czas na nieco definicji i nazewnictwa (imiona użyte w przykładach są przypadkowe!).

Podstawowymi pojęciami w plikach xml są - rodzic i dziecko. Rodzic to element obejmujący dziecko(!) - tak jak w poniższym przykładzie:

```
  <rodzic>
    <dziecko name="Jaś"/>
    <dziecko name="Małgosia"/>
  </rodzic>
```

Przy czym rodzic może mieć wiele dzieci, natomiast dziecko, tylko jednego rodzica (tym świat xmla różni się od świata rzeczywistego...).

Natomiast rodzic wszystkich rodziców to korzeć (ang. root) - i jako taki może być tylko jeden w dokumencie. Poniżej korzeniem jest $< dziadek >$

```
<dziadek>
  <rodzic name="Judyta">
    <dziecko name="Jaś"/>
    <dziecko name="Małgosia"/>
  </rodzic>
  <rodzic name="Grzegorz">
    <dziecko name="Staś"/>
    <dziecko name="Małgosia"/>
  </rodzic>
</dziadek>
```

Próba pracy z xmlem zawierającym wiele korzeni oznacza, że ktoś mógł źle sformatować ów plik, a skutkować to będzie komplikacjami przy użyciu narzędzi do pracy z xmlami. Poniżej niepoprawnie sformatowany plik xml:

```
<!-- Taki xml jest niepoprawny - są w nim dwa korzenie! -->
<rodzic name="Judyta">
  <dziecko name="Jaś"/>
  <dziecko name="Małgosia"/>
</rodzic>
<rodzic name="Grzegorz">
  <dziecko name="Staś"/>
  <dziecko name="Małgosia"/>
</rodzic>
```

Dodatkowo każdy element może mieć właśne atrybuty - w powyższych przykładach takim atrybutem był *name*. Jednocześnie elementy o tej samej nazwie wcale nie muszą mieć tych samych atrybutów

```
<!-- W xmlu elementy mogą mieć tę samą nazwę -->
<!-- ale inne atrubuty!                      -->
<rodzic name="Judyta">
  <dziecko name="Jaś"/>
  <dziecko name="Małgosia" surname="Iksińska"/>
</rodzic>
```

Z powyższego powodu xml jest bardzo dobrym formatem do trzymania danych o nieregularnej strukturze, które na przykład trudno ująć w tabeli. Równocześnie xml może być przydatny do przygotowywania wszelkiej maści konfigów, czy zapisywania modeli statystycznych (patrz https://en.wikipedia.org/wiki/Predictive_Model_Markup_Language). Oczywiście są też inne języki znaczników *yaml*, czy *JSON*.

Poszczególne elementy, oprócz atrybutów i dzieci, mogą mieć wartości tekstowe:

```
<!-- Jaś jest w tym przypadku wartością -->
<dziecko>Jaś</dziecko>
```

I na koniec rzecz która powinna być na początku:

```
<!-- Tak. To jest komentarz w xmlu -->
```

I to w zasadzie tyle najważniejszej terminologii xml'a! Pora na przejście do R.

Wczytujemy xml'a do R

Do obróbki xml'a najlepiej wykorzystywać pakiet xml2 napisany przez Hadley'a Wickhama. Samo nazwisko autora oznacza że pakiet jest bardzo prosty i przemyślany, a do tego prosty w obsłudze (jeżeli ktoś korzysta z R i nie jest zaznajomiony z dorobkiem Wickhama, powinien szybko nadrobić zaległości!).

I najważniejsze aktualna wersja xml2 znajdująca się na CRAN (0.1.2.9000) ma problem z kodowaniem polskich znaków - tzn. pojawiają się krzaczki - na szczęście w wersji deweloperskiej problem został już usunięty (ufff) - z tego też powodu - póki poprawki nie pojawią się na CRANie należy zainstalować wersję z githuba:

```
# Oczywiscie trzeba miec zainstalowane Rtools (na windowsie)
# https://cran.r-project.org/bin/windows/Rtools/
# Uzytkownicy linuxa dadzą radę:)
library(devtools)
install_github("hadley/xml2")
```

Poniżej przykładowy kawałek kodu:

```
library(xml2)

xmlText = '<?xml version="1.0" encoding="UTF-8"?>
<model>
  <!-- Opis xml modelu regresji liniowej -->
  <coefficients>
    <coeff name="Intercept" value="20.53" sd="11.01" intercept="true"/>
    <coeff name="price" value="30.1" sd="2.06" signif="true"/>
  </coefficients>
  <sd value="5.43"/>
  <rsqr>0.45</rsqr>
</model>'

# xml'a można wczytać ze stringa
xmlFromText = read_xml(xmlText)

# jak rowniez z pliku, xml2 sam zrozumie z czym ma do czynienia
cat(xmlText, file = "xmlTmp.xml")
xmlFromFile = read_xml("xmlTmp.xml")
```

Bardzo częstym problemem przy wczytywaniu xmli jest fakt, że nie zawsze są one dobrze sformatowane - na przykład - próba wczytania poniższego xml'a zakończy się błędem:

```
xmlTextErr = '<?xml version="1.0" encoding="UTF-8"?>
<model>
  <coefficients>
    <coeff name="Intercept" value="20.53" sd="11.01" intercept="true"/>
  </coefficients>
</model>
```

```
<model>
  <coefficients>
    <coeff name="Intercept" value="34.53" sd="13.01" intercept="true"/>
  </coefficients>
</model>
'

read_xml(xmlTextErr)
```

```
## Error in eval(expr, envir, enclos): Extra content at the end of the document [5]
```

Komunikat o błędzie nie jest zbyt jasny, ale parę sekund googlowania pozwoli znaleźć odpowiedź, co jest właściwym powodem błędu - w powyższym dokumencie nie ma korzenia - czyli znacznika obejmującego cały dokument. Poniżej poprawiona wersja (należało dodać $<models>$ na początku i $</models>$ na końcu dokumentu):

```
xmlTextCor = '<?xml version="1.0" encoding="UTF-8"?>
<models>
  <model>
    <coefficients>
      <coeff name="Intercept" value="20.53" sd="11.01" intercept="true"/>
    </coefficients>
  </model>
  <model>
    <coefficients>
      <coeff name="Intercept" value="34.53" sd="13.01" intercept="true"/>
    </coefficients>
  </model>
</models>
'

read_xml(xmlTextCor)
```

```
## {xml_document}
## <models>
## [1] <model>\n    <coefficients>\n      <coeff name="Intercept" value="20 ...
## [2] <model>\n    <coefficients>\n      <coeff name="Intercept" value="34 ...
```

Podstawowe operacje

By efektywnie korzystać z plików xml należy nauczyć się języka XPaht. Szerszy jego opis z przykładami można znaleźć między innymi na stronie http://www.w3schools.com/xsl/xpath_syntax.asp. W tym miejscu przedstawione zostanie jedynie podstawowe wyszukiwanie węzłów - jednak składnia XPath jest na tyle prosta, iż poznanie się na tyle by móc z niej efektywnie korzystać jest kwestią kilku minut.

```
# Jak zawsze dane
library(xml2)
xmlText = '<?xml version="1.0" encoding="UTF-8"?>
<model>
  <!-- Opis xml modelu regresji liniowej -->
  <coefficients>
```

```
    <coeff name="Intercept" value="20.53" sd="11.01" intercept="true"/>
    <coeff name="price" value="30.1" sd="2.06" signif="true"/>
  </coefficients>
  <sd value="5.43"/>
  <rsqr>0.45</rsqr>
</model>'
xmlAll = read_xml(xmlText)
```

```
# //nazwa - szukaj w calym drzwie
xml_find_all(xmlAll, "//coeff")
```

```
## {xml_nodeset (2)}
## [1] <coeff name="Intercept" value="20.53" sd="11.01" intercept="true"/>
## [2] <coeff name="price" value="30.1" sd="2.06" signif="true"/>
```

```
# /sciezka//nazwa - szukaj w drzewie ponizej sciezki
xml_find_all(xmlAll, "/model/coefficients//coeff")
```

```
## {xml_nodeset (2)}
## [1] <coeff name="Intercept" value="20.53" sd="11.01" intercept="true"/>
## [2] <coeff name="price" value="30.1" sd="2.06" signif="true"/>
```

```
# rsqr nie znajduje sie w coefficients wiec nie
# zostanie znalezione
xml_find_all(xmlAll, "/model/coefficients//rsqr")
```

```
## {xml_nodeset (0)}
```

Po pobraniu zestawu węzłów można w zasadzie wykonać dwie główne operacje - pobrać wartość znajdującą się między tagami (przypadek $< tag > wartość < /tag >$), lub pobrać wartość atrybutu (przypadek $< tagattr = "wartość" / >$).

```
# wartosc
rsqr = xml_find_all(xmlAll, "//rsqr")
xml_text(rsqr)
```

```
## [1] "0.45"
```

```
# atrybut
coeff = xml_find_all(xmlAll, "//coeff")
xml_attr(coeff, "value")
```

```
## [1] "20.53" "30.1"
```

```
# wszystkie atrybuty:
xml_attrs(coeff)
```

```
## [[1]]
##        name       value          sd   intercept
## "Intercept"     "20.53"     "11.01"      "true"
##
## [[2]]
##    name   value      sd  signif
## "price"  "30.1"  "2.06"  "true"
```

Przykłady praktycznego użycia

W przydkładach wykorzystywany jest plik tadeusz.xml, będący rezultatem działania programu TaKIPI 1.8. Plik powinien być w repozytorium w którym znajduje się niniejszy tutorial.

Chmura tagów z Pana Tadeusza:

W przykładzie używam tylko losowych 2000 słów inaczej chmura dłuugo się liczy.

```
library(wordcloud)
library(xml2)
library(magrittr)
xmlAll   = read_xml("tadeusz.xml")

# znalezienie wszystkich slow i pobranie ich z xml'a
allWords = xml_find_all(xmlAll, "//orth") %>% xml_text()

# wybranie slow o dlugosci > 3 litery
allWords = allWords[nchar(allWords) > 3]
wordcloud(sample(allWords,2000))
```

Pobranie wszystkich czasowników:

Pobranie wszystkich czasowników załatwić przy pomocy jednego zapytania xpath (tutaj też objawia się jego potęga):

```
library(xml2)
library(magrittr)
xmlAll   = read_xml("tadeusz.xml")
praets = xml_find_all(xmlAll, "/root//tok[contains(lex/ctag, 'praet')]/orth")
xml_text(praets) %>% head
```

```
## [1] "stracił"   "powróciła" "podniosł"  "mogł"      "Stał"      "Świeciły"
```

Najważniejszą częścią jest $tok[contains(lex/ctag,'praet')]$, w którym $contains(lex/ctag,'praet')$ jest warunkiem logicznym - sprawdzającym czy w węźle $lex/ctag$ znajduje się wartość $'praet'$. Jeżeli jest - wtedy z węzła tok zwracany jest węzeł $orth$ zawierający bazowe słowo.