assign4

May 20, 2020

1 Assignment 4

[7]:
```
import numpy as np
import pandas as pd
from lxml import html
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
```

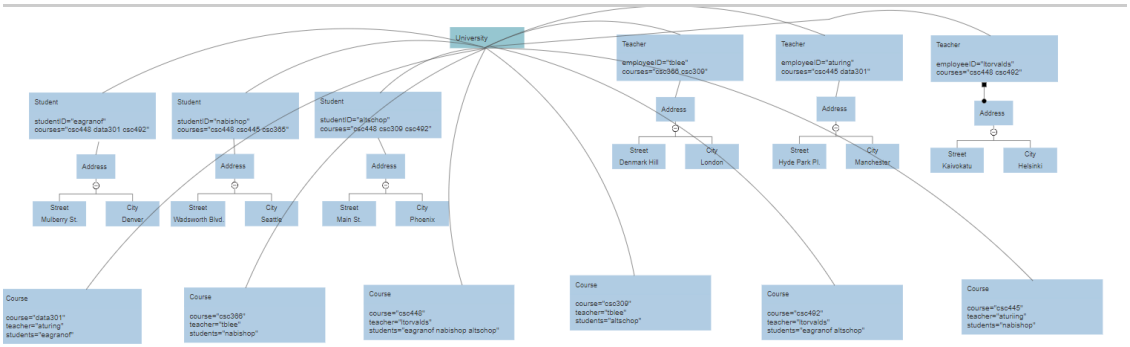
```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Ethan\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

Consider the following DTD:

```
<!DOCTYPE University [
    <!ELEMENT University (Student*, Teacher*, Course*)>
    <!ELEMENT Student (Name, Address+)>
        <!ATTLIST Student
            studentID ID #REQUIRED
            courses IDREFS #IMPLIED
        >
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT Address (Street, City)>
    <!ELEMENT Street (#PCDATA)>
    <!ELEMENT City (#PCDATA)>
    <!ELEMENT Teacher (Name, Address+)>
        <!ATTLIST Teacher
            employeeID ID #REQUIRED
            courses IDREFS #IMPLIED
        >
    <!ELEMENT Course (Name)>
        <!ATTLIST Course
            course ID #REQUIRED
            teacher IDREFS #REQUIRED
            students IDREFS #REQUIRED
        >
]>
```

1.1 Question 1

Draw an example XML tree for the DTD. Include several courses, teachers, and students (remember that nodes cannot share IDs).



1.2 Question 2

Show the XML document for the tree.

```
<?xml version="1.0" ?>
<University>
    <Student studentID="eagranof" courses="csc448 data301 csc492">
        <Name>Ethan Agranoff</Name>
        <Address>
            <Street>Mulberry St.</Street>
            <City>Denver</City>
        </Address>
    </Student>
    <Student studentID="nabishop" courses="csc448 csc445 csc366">
        <Name>Nick Bishop</Name>
        <Address>
            <Street>Wadsworth Blvd.</Street>
            <City>Seattle</City>
        </Address>
    </Student>
    <Student studentID="altschop" courses="csc448 csc309 csc492">
        <Name>Alex Tschopp</Name>
        <Address>
            <Street>Main St.</Street>
            <City>Phoenix</City>
        </Address>
    </Student>
    <Teacher employeeID="tblee" courses="csc366 csc309">
        <Name>Tim Berners-Lee</Name>
        <Address>
            <Street>Denmark Hill</Street>
            <City>London</City>
        </Address>
```

```
    </Teacher>
    <Teacher employeeID="ltorvalds" courses="csc448 csc492">
        <Name>Linus Torvalds</Name>
        <Address>
            <Street>Kaivokatu</Street>
            <City>Helsinki</City>
        </Address>
    </Teacher>
    <Teacher employeeID="aturing" courses="csc445 data301">
        <Name>Alan Turing</Name>
        <Address>
            <Street>Hyde Park Pl.</Street>
            <City>Manchester</City>
        </Address>
    </Teacher>
    <Course course="csc448" teacher="ltorvalds" students="eagranof nabishop altschop"></Course>
    <Course course="csc309" teacher="tblee" students="altschop"></Course>
    <Course course="csc366" teacher="tblee" students="nabishop"></Course>
    <Course course="csc492" teacher="ltorvalds" students="eagranof altschop"></Course>
    <Course course="csc445" teacher="aturing" students="nabishop"></Course>
    <Course course="data301" teacher="aturing" students="eagranof"></Course>
</University>
```

1.3 Question 3

Show the xpath query that returns the IDs of the students that are taking the course with ID=‘csc366’.

```
//Student[contains(@courses,"csc366")]/@studentID
```

[6]:
```
with open('uni.xml','r') as f:
    tree = html.fromstring(f.read())
csc366 = tree.xpath('')
print(*csc366)
```

```
nabishop
```

1.4 Question 4

Create a JSON file for the example in Q2.

```
{
    "University": {
        "Student": [
            {
                "studentID": "eagranof",
                "courses": "csc448 data301 csc492",
                "Name": "Ethan Agranoff",
                "Address": {
                    "Street": "Mulberry St.",
```

```
            "City": "Denver"
        }
    },
    {
        "studentID": "nabishop",
        "courses": "csc448 csc445 csc366",
        "Name": "Nick Bishop",
        "Address": {
            "Street": "Wadsworth Blvd.",
            "City": "Seattle"
        }
    },
    {
        "studentID": "altschop",
        "courses": "csc448 csc309 csc492",
        "Name": "Alex Tschopp",
        "Address": {
            "Street": "Main St.",
            "City": "Phoenix"
        }
    }
],
"Teacher": [
    {
        "employeeID": "tblee",
        "courses": "csc365 csc309",
        "Name": "Tim Berners-Lee",
        "Address": {
            "Street": "Denmark Hill",
            "City": "London"
        }
    },
    {
        "employeeID": "ltorvalds",
        "courses": "csc448 csc492",
        "Name": "Linus Torvalds",
        "Address": {
            "Street": "Kaivokatu",
            "City": "Helsinki"
        }
    },
    {
        "employeeID": "aturing",
        "courses": "csc445 data301",
        "Name": "Alan Turing",
        "Address": {
            "Street": "Hyde Park Pl.",
            "City": "Manchester"
```

```
            }
        }
    ],
    "Course": [
        {
            "course": "csc448",
            "teacher": "ltorvalds",
            "students": "eagranof nabishop altschop"
        },
        {
            "course": "csc309",
            "teacher": "tblee",
            "students": "altschop"
        },
        {
            "course": "csc366",
            "teacher": "tblee",
            "students": "nabishop"
        },
        {
            "course": "csc492",
            "teacher": "ltorvalds",
            "students": "eagranof altschop"
        },
        {
            "course": "csc445",
            "teacher": "aturing",
            "students": "nabishop"
        },
        {
            "course": "data301",
            "teacher": "aturing",
            "students": "eagranof"
        }
    ]
    }
}
```


1.5 Question 5

Create the query from Q3 for the JSON file (JSONPath)

```
$..Student[?(@.courses =~ csc366)].studentID
```

1.6 Question 6

Suppose that we have only three words in the language: a, b, and c, the documents:

d1: a a b a c a

d2: b b a a c c a

d3: a c c

and the query a b

a) Show the text vectors for the query and the documents.

[43]:
```
docs = {'d1':'a a b a c a', 'd2':'b b a a c c a', 'd3':'a c c','q':'a b'}
df = pd.DataFrame(data=docs,index=['text']).T.reset_index()
df.columns = ['document','text']
df['words'] = df.text.str.strip().str.split('[\W]+')
result = []
for i in range(0,len(df)):
    for word in df.iloc[i]['words']:
        result.append((df.iloc[i]['document'], word.lower()))
words = pd.DataFrame(result, columns=['document','word'])
counts = words.groupby('document')['word'].value_counts().to_frame().
 ↪rename(columns={'word':'frequency'})
counts
```

[43]:

		frequency
document	word	
d1	a	4
	b	1
	c	1
d2	a	3
	b	2
	c	2
d3	c	2
	a	1
q	a	1
	b	1

b) Normalize the vectors using the TF-IDF formulas. Make sure to use the 0.5 formula for the query.

[44]:
```
max_frequency = counts.groupby('document').max().rename(columns={'frequency':
 ↪'maxFreq'})
tf = counts.join(max_frequency)
tf['tf'] = tf['frequency'] / tf['maxFreq']
doc_count = df['document'].nunique()
doc_freq = words.groupby('word')['document'].nunique().to_frame().
 ↪rename(columns={'document':'df'})
doc_freq['idf'] = np.log2(doc_count / doc_freq['df'].values)
result = tf.join(doc_freq)
result['tfidf'] = result['tf'] * result['idf']
query = result.loc['q']
query['tfidf'] = (.5 + .5 * query['tf']) * query['idf']
```

```
result.loc['q']['tfidf'] = query['tfidf']
result
```

[44]:

document	word	frequency	maxFreq	tf	df	idf	tfidf
d1	a	4	4	1.000000	4	0.000000	0.000000
	b	1	4	0.250000	3	0.415037	0.103759
	c	1	4	0.250000	3	0.415037	0.103759
d2	a	3	3	1.000000	4	0.000000	0.000000
	b	2	3	0.666667	3	0.415037	0.276692
	c	2	3	0.666667	3	0.415037	0.276692
d3	c	2	2	1.000000	3	0.415037	0.415037
	a	1	2	0.500000	4	0.000000	0.000000
q	a	1	1	1.000000	4	0.000000	0.000000
	b	1	1	1.000000	3	0.415037	0.415037

c) Compute the cosine distance between the query and each of the documents using the normalized values.

[48]:
```
q = result.loc['q']
def cosine_similarity(d):
    q,d = q['tdidf'],d['tdidf']
    dot_prod = np.dot(q,d)
    q_scal = np.sqrt(q^2 + q^2)
    d_scal = np.sqrt(d^2 + d^2)
    return dot_prod / (q_scal * d_scal)

docs = result.loc['d1':'d3']
docs.apply(cosine_similarity)
```

```
␣
↪---------------------------------------------------------------------------

UnboundLocalError                         Traceback (most recent call␣
↪last)

<ipython-input-48-a7315b0dc806> in <module>
  8
  9 docs = result.loc['d1':'d3']
---> 10 docs.apply(cosine_similarity)


␣
↪~\AppData\Local\Programs\Python\Python38\lib\site-packages\pandas\core\frame.
↪py in apply(self, func, axis, broadcast, raw, reduce, result_type, args,␣
↪**kwds)
6926             kwds=kwds,
```

```
      6927         )
   -> 6928         return op.get_result()
      6929
      6930     def applymap(self, func):


     ␣
↪~\AppData\Local\Programs\Python\Python38\lib\site-packages\pandas\core\apply.
↪py in get_result(self)
       184             return self.apply_raw()
       185
   --> 186         return self.apply_standard()
       187
       188     def apply_empty_result(self):


     ␣
↪~\AppData\Local\Programs\Python\Python38\lib\site-packages\pandas\core\apply.
↪py in apply_standard(self)
       290
       291         # compute the result using the series generator
   --> 292         self.apply_series_generator()
       293
       294         # wrap results


     ␣
↪~\AppData\Local\Programs\Python\Python38\lib\site-packages\pandas\core\apply.
↪py in apply_series_generator(self)
       319             try:
       320                 for i, v in enumerate(series_gen):
   --> 321                     results[i] = self.f(v)
       322                     keys.append(v.name)
       323             except Exception as e:


     <ipython-input-48-a7315b0dc806> in cosine_similarity(d)
       1 q = result.loc['q']
       2 def cosine_similarity(d):
   ----> 3     q,d = q['tdidf'],d['tdidf']
       4     dot_prod = np.dot(q,d)
       5     q_scal = np.sqrt(q^2 + q^2)


     UnboundLocalError: ("local variable 'q' referenced before assignment",␣
↪'occurred at index frequency')
```

[ ]: