Introduction to Web Science

Assignment 8

Prof. Dr. Steffen Staab
staab@uni-koblenz.de

René Pickhardt
rpickhardt@uni-koblenz.de

Korok Sengupta
koroksengupta@uni-koblenz.de

Olga Zagovora
zagovora@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science
University of Koblenz-Landau

Submission until: January 11, 2017, 10:00 a.m.
Tutorial on: January 13, 2017, 12:00 p.m.

Please look at all the lessons of part 2 in particular **Similarity of Text** and **graph based models**

For all the assignment questions that require you to write code, make sure to include the code in the answer sheet, along with a separate python file. Where screen shots are required, please add them in the answers directly and not as separate files.

Other than that this sheet is mainly designed to review and apply what you have learnt in part 2 it is a little bit larger but there is also more time over the x-mas break. In any case we wish you a mery x-mas and a happy new year.

Team Name: Echo

1 Similarity - (40 Points)

This assignment will have one exercise which is dived into four subparts. The main idea is to study once again the web crawl of the Simple English Wikipedia. The goal is also to review and aply your knowledge from part 2 of this course.

We have constructed two data sets from it which are all the articles and the link graph extracted from Simple English Wikipedia. The extracted data sets are stored in the file `http://141.26.208.82/store.zip` which contains a pandas container and can be read with pandas in python. In subsection "1.5 Hints" you will find some sample python code that demonstrates how to easily access the data.

With this data set you will create three different models with different similarity measures and finally try to evaluate how similar these models are.

This assignment requires you to handle your data in efficient data structures otherwise you might discover runtime issues. So please read and understand the full assignment sheet with all the tasks that are required before you start implementing some of the tasks.

1.1 Similarity of Text documents (10 Points)

1.1.1 Jaccard - Similarity on sets

1. Build the word sets of each article for each article id.
2. Implement a function `calcJaccardSimilarity(wordset1, wordset2)` that can calculate the jaccard coefficent of two word sets and return the value.
3. Compute the result for the articles `Germany` and `Europe`.

1.1.2 TF-IDF with cosine similarity

1. Count the term frequency of each term for each article
2. Count the document frequencies of each term.
3. For each article id provide a dictionary of terms occuring in the article together with their tf-idf scores as the corresponding values.
4. Implement a function `calculateCosineSimilarity(tfIdfDict1, tfIdfDict2)` that computes the cosine similarity for two sparse tf-idf vectors and returns the value.
5. Compute the result for the articles `Germany` and `Europe`.

Answer of 1.1(1.1.1,1.1.2) and 1.2 are given together in after 1.2

1.2 Similarity of Graphs (10 Points)

You can understand the similarity of two articles by comparing their sets of outlinks (and see how much they have in common). Feel free to reuse the `computeJaccardSimilarity` function from the first part of the exercise. This time do not aply it on the set of words within two articles but rather on the set of outlinks being used within two articles. Again compute the result for the articles `Germany` and `Europe`.

Answer

```
import pandas as pd
import numpy as np
import time
import math
import re

store = pd.HDFStore('store.h5')
df1 = store['df1']
df2 = store['df2']

### 1.1 Similarity of Text documents
## 1.1.1 Jaccard - Similarity on sets
# a function to create word sets for each text
def make_wordsets():
    return lambda x: map(lambda y: re.sub('[^a-zA-Z0-9\n\ ]', '', y.strip()), x)

## a function to return the Jaccard similarity for any two sets
def calcJaccardSimilarity(wordset1, wordset2):
    return len(set(wordset1).intersection(set(wordset2))) / float(len(set(wordset1).unio

# saving the created word sets in sets
df1['sets'] = df1.text.str.lower().str.split().apply(make_wordsets())

# text to lower case no special characters
df1['text'] = df1.text.str.lower().str.strip().apply(lambda x: re.sub('[^a-zA-Z0-9\n\ ]'

# print the Jaccard similarity for articles in loc[0] and loc[1], for example
print 'Jaccard similarity for article in loc[0] and article in loc[1]: ', calcJaccardSim

# print Jaccard similarity for Germany and Europe article
print 'Jaccard similarity for Germany and Europe article:', calcJaccardSimilarity(set(df

## 1.1.2 TF-IDF with cosine similarity
```

```
# a function to create a set of all unique words in the file
def makeoneset(dataframe):
    results = set()
    dataframe.apply(results.update)
    return results
aset = makeoneset(df1.sets)

# idf (here was the runtime issue that I ran into and I couldn't think of another way to
start_time = time.time()
dictionary = dict()
counter = 0
for word in list(aset):
    dictionary[word] = round(math.log(len(aset) / float(len(df1[df1.text.str.contains(wo
    counter += 1
    if counter == 1000:
        print("--- %s seconds ---" % (time.time() - start_time))
        break

# tf
def makeDict(setofwords):
    counts = dict()
    for i in setofwords:
      counts[i] = counts.get(i, 0) + 1
    return counts

# saving tf scores as a dictionary for each document
df1['tf'] = df1.sets.apply(lambda x: makeDict(x))
print 'TF example for article loc[0]: ', df1.loc[0].tf

# function to calculate tfidf (takes in two dictionaries)
def tfidf(x, y):
    return dict((k, v * x[k]) for k, v in y.items() if k in x)

# saving tfidf dictionary
df1['tfidf'] = df1.sets.apply(lambda x: tfidf(makeDict(x), dictionary))
print 'TFIDF example for article in loc[0]: ', df1.loc[0].tfidf

# fix the shape of the series for multiplication (document vectors make same size with a
def fixShape(doc):
    if doc.size != 10:
        for i in xrange(abs(doc.size-len(dictionary))): # len(aset)
           doc = doc.set_value(i, i*0)
    return doc
```

```
# a function to get the similarity of two dictionaries
def dicOneMultiDicTwo(dictt1, dictt2):
    return sum(fixShape(pd.Series(dictt1)).values * fixShape(pd.Series(dictt2)).values)

# a function to find the length of the vector
def lengthofavector(dict11):
    return round(math.sqrt(sum((val**2) for val in dict11.values())), 2)

# a function to calculate Cosine Similarity
def calculateCosineSimilarity(tfIdfDict1, tfIdfDict2):
    print('Cosine similarity between {} and {}'.format(tfIdfDict1, tfIdfDict2))
    return round(dicOneMultiDicTwo(tfIdfDict1, tfIdfDict2) / (lengthofavector(tfIdfDict1

print calculateCosineSimilarity(df1[df1.name=="Europe"].tfidf.item(), df1[df1.name=="Germ

### 1.2 Similarity of Graphs
print 'Jaccard similarity for Germany and Europe out links:', calcJaccardSimilarity(df2[d

store.close()
```

```
D:\MScWebScience\IntroToWS\assignments\Echo\Echo\assignment8_WorkingFolder>python task1p1and
p2.py
Jaccard similarity for article in loc[0] and article in loc[1]:  0.108695652174
Jaccard similarity for Germany and Europe article: 0.046178343949
--- 53.6570000648 seconds ---
TF example for article loc[0]:  {'who': 2, 'german': 3, 'it': 3, 'one': 1, 'talking': 1, 'as
': 1, 'himself': 1, 'in': 2, 'thinks': 1, 'different': 1, 'from': 1, 'area': 1, 'federal': 1
, 'things': 1, 'when': 1, 'same': 1, 'also': 1, 'republic': 1, 'someone': 2, 'earlier': 1, '
germany': 2, 'herself': 1, 'a': 2, 'about': 1, 'language': 1, 'countries': 1, 'of': 3, 'see'
: 1, 'person': 2, 'can': 4, 'country': 1, 'lives': 1, 'the': 5, 'or': 3, 'mean': 4}
TFIDF example for article in loc[0]:  {}
Cosine similarity between {} and {'': 2.98, 'czech': 6.91, 'travel': 5.27, 'had': 7.1, 'xvi'
: 7.38, 'want': 8.98, 'has': 23.04, 'bringing': 7.42}
nan
Jaccard similarity for Germany and Europe out links: 0.0621761658031
```

1.3 How similar have our similarities been? (10 Points)

Having implemented these three models and similarity measures (text with Jaccard, text with cosine, graph with Jaccard) our goal is to understand and quantify what is going on if they are used in the wild. Therefore in this and the next subtask we want to try to give an answere to the following questions.

- Will the most similar articles to a certain article always be the same independent which model we use?
- How similar are these measures to each other? How can you statistically compare them?

Assume you could use the similarity measure to compute the top k most similar articles for each article in the document collection. We want to analyze how different the rankings for these various models are.

Do some research to find a statistical measure (either from the lectures of part 2 or by doing a web search and coming up with something that we haven't discussed yet) that could be used best to compare various rankings for the same object.

Explain in a short text which measure you would use in such an experiment and why you think it is usefull for our task.

Answer
Here we have to calculate 3 different ranking depending on what these 3 different models give result on a certain condition on a same set of data.
We can take different approach like calculate the text similarities or calculate based on intersections among the rankings. But after little study what has been found is little more reliable and easier approach for comparing the rankings, it is 'Set Based Measure - Rank Biased Overlap'. This measure is used to compare various rankings for the same object.
Here what we are going to do is,
1. Take a random article x.
2. Take a set (y(100)) of 100 article, some from before the index and some from after the index of x.
3. Compare each element from set y(100) with x.
4. Sort the Jaccard, Cosine similarity and the graph with Jaccard value and take top 10 most similar articles for each similarty measure js(10), cs(10), jgs(10).
5. Implement 'Set Based Measure - Rank Biased Overlap' on these 3 different rankings. We will find if they are same or if different, how different are they. We will also find which of the 2 rankings are more similar.
Here is a the reference links where I found about this 'Set Based Measure - Rank Biased Overlap':
https://ragrawal.wordpress.com/2013/01/18/comparing-ranked-list
https://github.com/ragrawal/measures
Other Links:
https://en.wikipedia.org/wiki/Rank_correlation
http://www.williamwebber.com/research/papers/wmz10_tois.pdf

After calculating this we will be able to answer the question asked in the task 1.3 (According to our choosen methodology - Set Based Measure - Rank Biased Overlap.).

1.4 Implement the measure and do the experiment (10 Points)

After you came up with a measure you will most likely run into another problem when you plan to do the experiment.

Since runtime is an issue we cannot compute the similarity for all pairs of articles. Tell us:

1. How many similarity computations would have to be done if you wished to do so?
2. How much time would roughly be consumed to do all of these computations?

A better strategy might be to select a couple of articles for which you could compute your meassure. One strategy would be to select the 100 longest articles. Another strategy might be to randomly select 100 articles from our corpus.

Computer your three similarity measures and evaluate them for these two strategies of selecting test data. Present your results. Will the results depend on the method for selecting articles? What are your findings?

1.5 Hints:

1. In oder to access the data in python, you can use the following pice of code:

   ```
   import pandas as pd
   store = pd.HDFStore('store.h5')
   df1=store['df1']
   df2=store['df2']
   ```

2. Variables df1 and df2 are pandas DataFrames which is tabular data structure. df1 consists of article's texts, df2 represents links from Simple English Wikipedia articles. Variables have the following columns:
   - "name" is a name of Simple English Wikipedia article,
   - "text" is a full text of the article "name",
   - "out_links" is a list of article names where the article "name" links to.
3. In general you might want to store the counted results in a file before you do the similarity computations and all the research for the third and fourth subtask. Doing all this counting and preperation might allready take quite some runtime.
4. When computing the sparse tf-idf vectors you might allready want to store the eukleadan length of the vectors. otherwise you might discover runtime issues when computing the length again for each similarity computation.
5. Finding the top similar articles for a given article id requires you to compute the similarity of the given article with comparison to all the other known articles and

extract the top 5 similarities. Bare in mind that these are quite a lot of similarity computations! You can expect a runtime to find the top similar articles with respect to one of the methods to be up to 10 seconds. If it takes significant longer then you probably have not used the best data structures handle your data.

6. **Even though many third party libraries exist to do this task with even less computational effort those libraries must not be used.**
7. You can find more information about basic usage of pandas DataFrame in pandas documentation.
8. Here are some usefull examples of operations with DataFrame:

```
import pandas as pd

store = pd.HDFStore('store.h5')#read .h5 file
df1=store['df1']
df2=store['df2']
print df1['name'] # select column "name"
print df1.name # select column "name"
print df1.loc[9] #select row with id equals 9
print df1[5:10] #select rows from 6th to 9th (first row is 0)
print df2.loc[0].out_links #select outlinks of article with id=0

#show all columns where column "name" equals "Germany"
print df2[df2.name=="Germany"]

#show column out_links for rows where name is from list ["Germany","Austria"]
print df2[df2.name.isin(["Germany","Austria"])].out_links

#show all columns where column "text" contains word "good"
print df1[df1.text.str.contains("good")]

#add word "city" to the beginning of each text value
#(IT IS ONLY SHOWS RESULT OF OPERATION, see explanation below!)
print df1.text.apply(lambda x: "city "+x)

#make all text lower case and split text by spaces
df1[["text"]]=df1.text.str.lower().str.split()

def do_sth(x):
        #here is your function
        #
        #
        return x
```

```
#apply do_sth function to text column
#Iit will not change column itself, it will only show the result of aplication
print df1.text.apply(do_sth())

#you always have to assign result to , e.g., column,
#in order it affects your data.
#Some functions indeed can change the DataFrame by
#applying them with argument inplace=True
df1[["text"]]=df1.text.apply(do_sth())

#delete column "text"
df1.drop('text', axis=1, inplace=True)
```

Important Notes

Submission

- Solutions have to be checked into the github repository. Use the directory name `groupname/assignment8/` in your group's repository.
- The name of the group and the names of all participating students must be listed on each submission.
- Solution format: all solutions as *one* PDF document. Programming code has to be submitted as Python code to the github repository. Upload *all* `.py` files of your program! Use `UTF-8` as the file encoding. *Other encodings will not be taken into account!*
- Check that your code compiles without errors.
- Make sure your code is formatted to be easy to read.
  - Make sure you code has consistent indentation.
  - Make sure you comment and document your code adequately in English.
  - Choose consistent and intuitive names for your identifiers.
- Do *not* use any accents, spaces or special characters in your filenames.

Acknowledgment

This latex template was created by Lukas Schmelzeisen for the tutorials of "Web Information Retrieval".

LaTeX

Currently the code can only be build using LuaLaTeX, so make sure you have that installed. If on Overleaf, there's an error, go to settings and change the LaTeXengine to `LuaLaTeX`.