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: QUEBEC

1. Daniel Kostic
2. Stefan Vujovic
3. Igor Fedotov

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:

1.1.1

First, we have built the word sets for each article id:

```
import re
df1["words"] = df1.text.apply(lambda x: set(re.findall("\w+",str(x))))
df1["word_list"] = df1.text.apply(lambda x: re.findall("\w+",str(x)))
```

The output looks like this:

```
In [8]: df1.head()
```

Out[8]:

	name	text	words	word_list
0	German	**German** can mean different things. It can ...	{same, someone, himself, mean, When, one, also...	[German, can, mean, different, things, It, can...
1	Coat_of_Arms_of_Germany_0bb4	The **coat of arms of Germany** (German **Wapp...	{to, _Schwarz, religious, coat, was, god, thro...	[The, coat, of, arms, of, Germany, German, Wap...
2	Flag_of_Germany_0658	The **flag of Germany** (German: _Bundesflagge...	{out, form, new, _Bundesflagge_, was, colours,...	[The, flag, of, Germany, German, _Bundesflagge...
3	Das_Lied_der_Deutschen_3057	**Das Lied der Deutschen** (The Song of the Ge...	{Hoffman, known, Einigkeit, British, Siegerkra...	[Das, Lied, der, Deutschen, The, Song, of, the...
4	Map	A **map** is usually a picture of the Earth or...	{make, screen, also, Earth, only, one, If, The...	[A, map, is, usually, a, picture, of, the, Ear...

Then we implemented a function for calculating the Jaccard coefficient for two word sets:

```
def calc_jaccard_simmilarity(wordset1, wordset2):
    intersetion_cardinality = len(set.intersection(wordset1, wordset2))
    union_cardinality = len(set.union(wordset1, wordset2))
    return intersetion_cardinality/float(union_cardinality)
```

When we passed the word sets for articles ”Germany” and ”Europe” into this function, it returned this:

```
In [14]: calc_jaccard_simmilarity(europe_words, germany_words)
Out[14]: 0.043219076005961254
```

1.1.2

We calculated the term frequency of each term for each article and added it into our DataFrame:

```
from collections import Counter
df1['tf'] = df1.word_list.apply(lambda x: Counter(x))
```

This is how we got the term frequency and tf-idf:

```
words = df1.words.tolist()

all_words = [word for sets in words for word in sets]

doc_freq = Counter(all_words)


def tf_dict_to_tfidf(tf_dic):
    new_dic = {}
```

```
    for k,v in tf_dic.items():
        new_dic[k] = v / float(doc_freq[k])
    return new_dic
df1['tf_idf'] = df1.tf.apply(tf_dict_to_tfidf)
```

We implemented a function for calculating the cosine similarity:

```
from math import sqrt
def euclidian_len(d1):
    return sqrt(sum([v*v for k,v in d1.items()]))

def scalar(d1, d2):
    suma = 0
    for k,v in d1.items():
        try:
            suma += v * d2[k]
        except KeyError:
            pass
    return suma

def calc_cosine_similarity(d1,d2):
    return (scalar(d1,d2)/(euclidian_len(d1)*euclidian_len(d2)))
```

The cosine similarity for articles ”Europe” and ”Germany” is:

```
In [26]: calc_cosine_similarity(dfg.tf_idf.values[0], dfe.tf_idf.values[0])
Out[26]: 0.0002954073047552391
```

1.2 Similarity of Graphs (10 Points)

You can understand the similarity of two articles by comparing their sets of out links (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 apply it on the set of words within two articles but rather on the set of out links being used within two articles. Again compute the result for the articles `Germany` and `Europe`.

1. Answer: First selecting Europe and Germany from data frames containing out links. After that, calc jaccard similarity can be applied just by turning lists of links to sets. The result is 0.9, which means that these Articles are very similar according to the out links.

In [27]:
```
dfe2 = df2[df2["name"] == "Europe"]
europe_links = dfe2.out_links.values[0]
dfg2 = df2[df2["name"] == "Germany"]
germany_links = dfg2.out_links.values[0]
```

In [28]:
```
df2.head()
```

Out[28]:

	name	out_links
0	German	[German_language, Country, Germany, Frank, Hol...
1	Coat_of_Arms_of_Germany_0bb4	[German_language, Germany, Flag_of_Germany_065...
2	Flag_of_Germany_0658	[German_language, Black, Red, Gold, Flag_of_Ge...
3	Das_Lied_der_Deutschen_3057	[Germany, Joseph_Haydn_fcd5, Hoffmann_von_Fall...
4	Map	[Earth, Photograph, Interpretation, Chart, Dra...

In [29]:
```
calc_jaccard_simmilarity(set(europe_links), set(germany_links))
```

Out[29]:
```
0.9130434782608695
```

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 answer 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?

Answer:

- To answer this question we compared all the articles to ”Germany”.

```
df1['cos_ger'] = df1.tf_idf.apply(lambda x: calc_cosine_similarity(x, dfg.tf_idf.values[0]))
```

```
df1['jac_ger'] = df1.words.apply(lambda x: calc_jaccard_simmilarity(x, germany_words))
```

```
df2['jac_ger_links'] = df2.out_links.apply(lambda x: calc_jaccard_simmilarity(set(x), set(germany_links)))
```

  After, we have merged all the results and sorted once according to one column, and taken the top 10 results.

```
df3 = pd.merge(df1[['name', 'cos_ger', 'jac_ger']], df2[['name', 'jac_ger_links']], on='name', how='left')
```

```
cos_text = df3.sort_values('cos_ger', ascending=False).head(10)['name']
jac_text = df3.sort_values('jac_ger', ascending=False).head(10)['name']
jac_links = df3.sort_values('jac_ger_links', ascending=False).head(10)['name']
```

When comparing the results, we can see that all the measures set Germany as the most similar article, which is to be expected. But all the other articles are different in each picture, so NO, most similar articles will not be the same after changing the model.

cos_text

27444	Germany
14680	Bringing_Up_Baby_1a24
26682	Zugspitze
4264	Neuendorf,_Switzerland_78d4
18929	Politics_of_Germany_8451
26903	Courts_of_Germany_6acc
27314	Enabling_Act_39b1
26582	Neuendorf-Sachsenbande_b48e
25851	Stefan_Zweig_d4fd
27341	Reichstag_(disambiguation)

jac_text

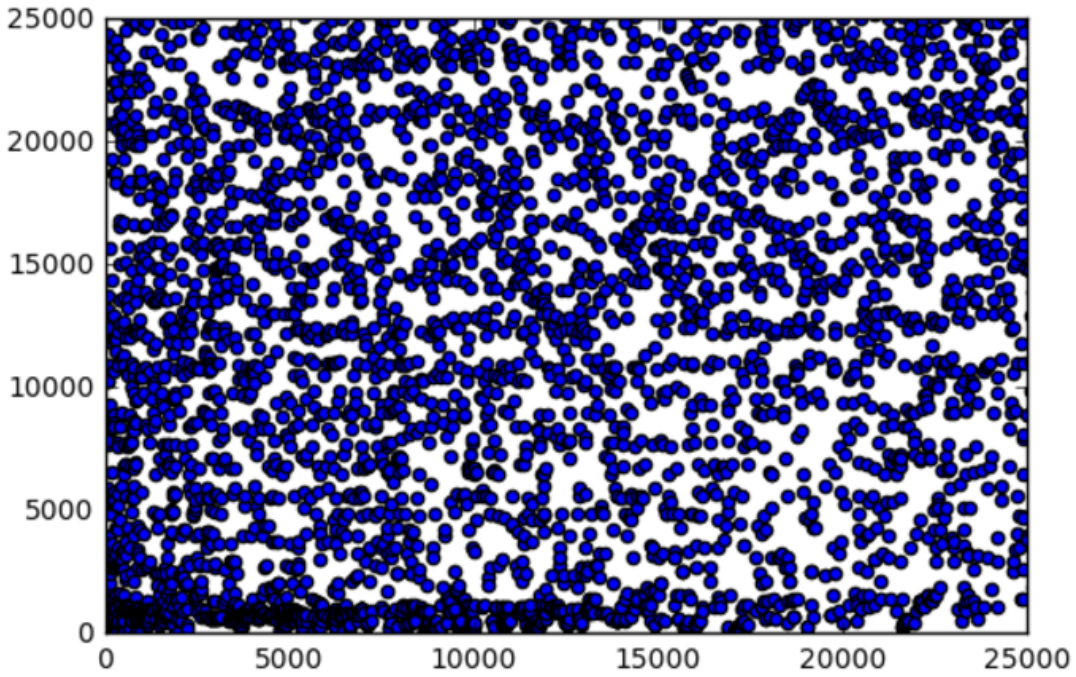
27444	Germany
27411	Prussia
27400	World_War_I_9429
18929	Politics_of_Germany_8451
33	Netherlands
29	Poland
27378	Treaty_of_Versailles_58e2
25780	Judaism
27383	Spain
26151	Finland

jac_links

27444	Germany
18929	Politics_of_Germany_8451
26613	Trinidad_and_Tobago_6d19
26742	Dominica
18620	List_of_cities_in_Germany_with_more_than_100,0...
26818	North_Rhine-Westphalia_29ea
26106	Estonia
34	Belgium
26112	Georgia_(country)
25384	Isle_of_Man_10ef

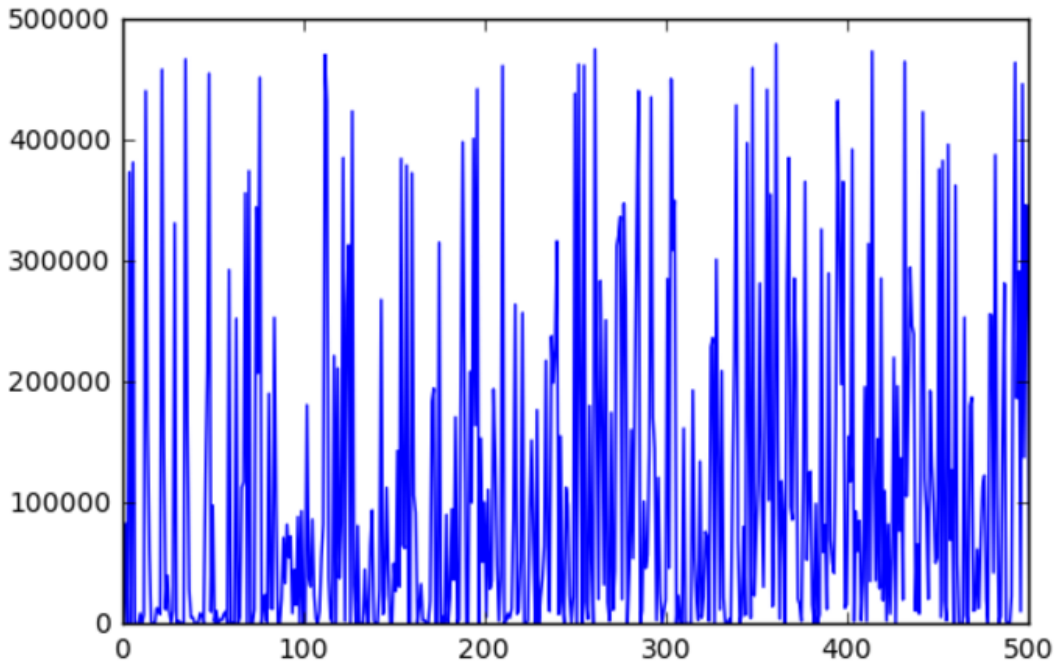
- How similar are these measures? How to compare them statistically? To answer this question we have chosen to implement the Spearman's rank correlation coefficient. Spearman's coefficient takes values between -1 and 1. If the ranks are very close, and very similar, the coefficient will be close to 1. If there is no correlation, the value will be 0. If the ranking are completely opposite one from another, the value will be -1.
  Spearman's correlation coefficient, measures the strength and direction of association between two ranked variables. Spearman's correlation determines the strength and direction of the monotonic relationship between your two variables rather than the strength and direction of the linear relationship between your two variables, which is what Pearson's correlation determines. That is, if a scatter plot shows that the relationship between your two variables looks monotonic you would run a

Spearman's correlation because this will then measure the strength and direction of this monotonic relationship.
After plotting our data it was still not really obvious is there a monotonic relationship in our data, but we decided to use it anyway.
Title: Looking at first 25000 results
X-axis: Cosine-text ranks (sorted)
Y-axis: Jaccard-text ranks

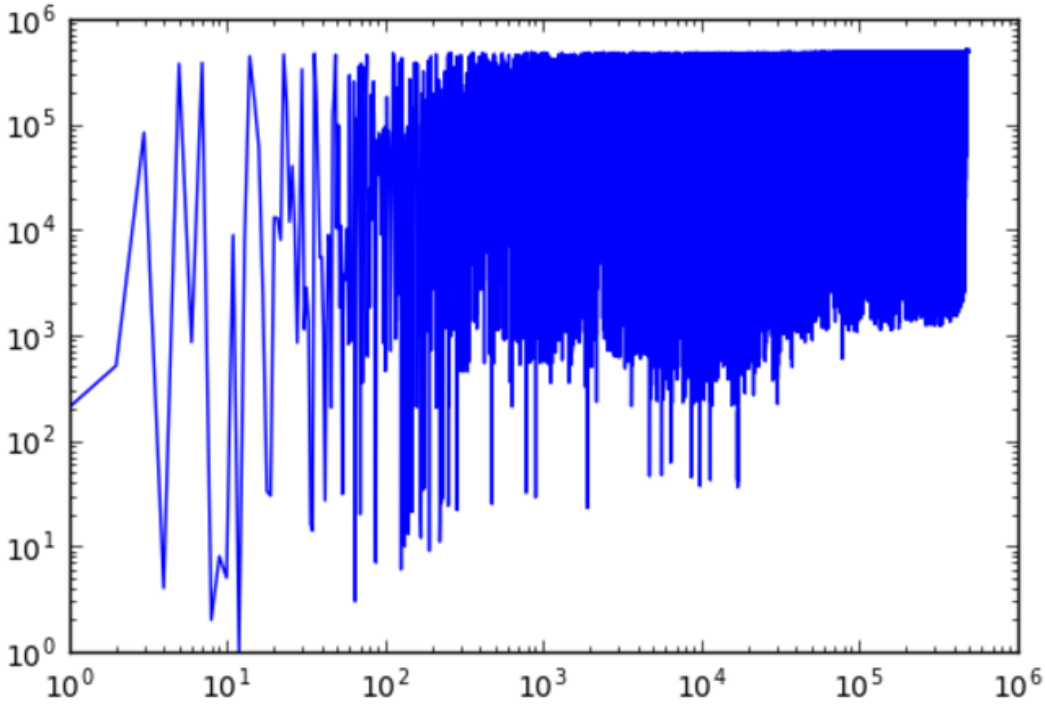


Looking at loglog plot, and the absolute rank difference of first 500 results gave us some insights.
Title: Absolut diff of 500 results - cosine and jaccard
X-axis: Comparison number
Y-axis: Absolute difference cosine and jaccard on text

```
import matplotlib.pyplot as plt
plt.plot(df_sim['diff_abs'])
plt.axis([0, 500, 0, 500000])
plt.show()
```



Loglog plot
Title: Log log plot of cosine and jaccard
X-axis: Cosine-text ranks (sorted)
Y-axis: Jaccard-text ranks



After realizing that we forgot to label our axes, we decided to apologize and write them in text due to time limitations.

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.

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?

Answer:

- In our case the number of similarity computations that would have to be computed equals the squared number of articles minus number of articles divided by 2. This is also the number of combinations when taking 2 elements.

```
(rows**2 - rows)/2
```

```
377918778.0
```

  For estimating the time of calculation we compared the speed of calculating a hundred randomly chosen and a hundred longest articles. When calculating the Jaccard coefficient the time needed was around 2s in case of top 100 longest articles, and around 900ms in case of a 100 random articles. While using the cosine similarity the time needed was around 700ms in case of top 100 longest articles, and around 500ms in case of a 100 random articles.
  In our implementation, 1,000,000 calculations are performed when calculating similarities for a 1000 articles, and that lasts for 15 seconds. As our data set has a number of 27493 articles, we will have 377918778 calculations(if we optimize the algorithm!). We can try to interpolate that for 377 million results it will take around 15*377 seconds, or 1,5 hours. It would last even longer as we would have more data to access, search etc.

- When selecting the longest 100 articles, by number of words, most measures are still very similar according to Spearman's coefficient, but the most similar ones are Cosine and Jaccard on text. It makes sense for these two to be similar as both

work with text data.

```
# cos and jaccard
rho(df_sim['diff'], df_sim.shape[0])
```

```
0.3545696681548253 8
```

```
# cos and jaccard_links
rho(df_sim['diff1'], df_sim.shape[0])
```

```
0.21080742988704948
```

```
# jaccard and jaccard_links
rho(df_sim['diff2'], df_sim.shape[0])
```

```
0.19285901607361966
```

When comparing 100 random articles, Jaccard on text and cosine are still the 2 most similar, but Jaccard on links becomes very different. We conclude this is because random articles probably have just a few links, which degrades the performance of this metric.

```
# cos and jaccard
rho(df_sim['diff'], df_sim.shape[0])
```

```
0.30117138866925242
```

```
# cos and jaccard_links
rho(df_sim['diff1'], df_sim.shape[0])
```

```
0.24478891605746378
```

```
# jaccard and jaccard_links
rho(df_sim['diff2'], df_sim.shape[0])
```

```
0.07782406466649249 8
```

1.5 Hints:

1. In order to access the data in python, you can use the following piece 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`.