Introduction to Web Science

Assignment 7

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: December 14, 2016, 10:00 a.m.
Tutorial on: December 16, 2016, 12:00 p.m.

Please look at the lessons 1) **Similarity of Text** & 2) **Generative 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.

Team Name: Oscar

1 Modeling Text in a Vector Space and calculate similarity (10 points)

Given the following three documents:

D_1 = this is a text about web science

D_2 = web science is covering the analysis of text corpora

D_3 = scientific methods are used to analyze webpages

1.1 Get a feeling for similarity as a human

Without applying any modeling methods just focus on the semantics of each document and decide which two Documents should be most similar. Explain why you have this opinion in a short text using less than 500 characters.

We think that Document 1 and Document 2 are most similar. We base our opinion on the fact that these two documents have more words in common than any other pair from the whole corpus of documents. The common words between these two documents are is, text, web, science. While, when we compare Document 1 with Document 3 and Document 2 with Document 3,we find no words in common for each case.

1.2 Model the documents as vectors and use the cosine similarity

Now recall that we used vector spaces in the lecture in order to model the documents.

1. How many base vectors would be needed to model the documents of this corpus?

   To model the documents of this corpus 19 base vectors would be needed.

2. What does each dimension of the vector space stand for?

   The dimensionality of the vector is the number of words in the vocabulary (the number of distinct words occurring in the corpus).

3. How many dimensions does the vector space have?

   Our vector space has 19 dimensions as there are 19 different terms in our corpus.

4. Create a table to map words of the documents to the base vectors.

5. Use the notation and formulas from the lecture to represent the documents as document vectors in the word vector space. You can use the term frequency of the words as coefficients. You can / should omit the inverse document frequency.

   vector (d1) = (1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

   vector (d2) = (0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)

Terms																		
this	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
is	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
text	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
about	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
web	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
science	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
covering	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
the	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
analysis	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
of	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
corpora	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
scientific	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
methods	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
are	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
used	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
to	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
analyze	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
webpages	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Terms	tf(D1)	tf(D2)	tf(D3)
this	1	0	0
is	1	1	0
a	1	0	0
text	1	1	0
about	1	0	0
web	1	1	0
science	1	1	0
covering	0	1	0
the	0	1	0
analysis	0	1	0
of	0	1	0
corpora	0	1	0
scientific	0	0	1
methods	0	0	1
are	0	0	1
used	0	0	1
to	0	0	1
analyze	0	0	1
webpages	0	0	1

vector (d3) = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1)

6. Calculate the cosine similarity between all three pairs of vectors.

   s < d1,d2 > = 0+1+0+1+0+1+1+0+0+0+0+0+0+0+0+0+0+0 = 4

   s < d1,d3 > = 0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0 = 0

   s < d2,d3 > = 0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0 = 0

7. According to the cosine similarity which 2 documents are most similar according to the constructed model.

   According to the cosine similarity, Document 1 and Document 2 are most similar.

1.3 Discussion

Do the results of the model match your expectations from the first subtask? If yes explain why the vector space matches the similarity given from the semantics of the documents. If no explain what the model lacks to take into consideration. Again 500 Words should be enough.

The results of the model match our expectations from the first subtask. One of the reasons that made the prediction easier was that Document 1 and Document 2 had no words in common with Documents 3. Also by checking the semantic of Document 1 and Document 2, we saw that they had 4 words in common. This makes these documents more similar to each other than to Document 3. The vector space matches the similarity given from the semantics of the document because the vector space uses the cosine similarity. This measure of similarity is more correct than Euclidian distance because it doesn't focus on the length of the vectors but in the angle that they form. The closer the angle, more similar the vectors are.

2 Building generative models and compare them to the observed data (10 points)

This week we provide you with two probability distributions for characters and spaces which can be found next to the exercise sheet on the WeST website. Also last week we provided you with a dump of Simple English Wikipedia which should be reused this week.

2.1 build a generator

Count the characters and spaces in the Simple English Wikipedia dump. Let the combined number be n. Use the sampling method from the lecture to sample n characters (which could be letters or a space) from each distribution. Store the result for the generated text for each distribution in a file.

2.2 Plot the word rank frequency diagram and CDF

Count the resulting words from the provided data set and from the generated text for each of the probability distributions. Create a word rank frequency diagram which contains all 3 data sets. Also create a CDF plot that contains all three data sets.

2.3 Which generator is closer to the original data?

Let us assume you would want to creat a test corpus for some experiments. That test corpus has to have a similar word rank frequency diagram as the original data set. Which of the two generators would you use? You should perform the Kolmogorov Smirnov test as discussed in the lecture by calculating the maximum pointwise distance of the CDFs.

How do your results change when you generate the two text corpora for a second or third time? What will be the values of the Kolmogorov Smirnov test in these cases?

2.4 Hints:

1. Build the cummulative distribution function for the text corpus and the two generated corpora
2. Calculate the maximum pointwise distance on the resulting CDFs
3. You can use `Collections.Counter`, `matplotlib` and `numpy`. You shouldn't need other libs.

The results i maintained for the maximum pointwise distance were 0.41 and 0.44 for zipfDist and naturalDist consequtively. This means that the zipf generator is better because it has a smaller max pointwise distance value (it's closer to the origianal corpus).

The more we generate corpuses, the more we move away from our original corpus no matter how good our generator is. So the value the Kolmogorov Smironov will keep on getting larger and larger the more we generate.

```
1: import random
2: import matplotlib.pyplot as plt
3: from collections import Counter
4: import numpy as np
5: import operator
6:
7: zipf = open('zipf.txt', 'w')
8: uniform = open('uniform.txt','w')
9:
10: #function for retreiving a character from a dictionary with a random flip
11: def retreive_char( dict ):
12:     random_number=random.random()
13:     output=' '
14:     for key, value in dict.items():
15:         if(value>=random_number):
16:             output=key
17:             break
18:     return output
19: #function to calculate the cdf just like the one in the lecture
20: def getFileCDF(file):
21:     c = Counter(file.read().split())
22:     words,frequencies = zip(*c.most_common())
23:     cumsum= np.cumsum(frequencies)
24:     normedcumsum = [x/float(cumsum[-1]) for x in cumsum]
25:     wrank = {words[i]:i+1 for i in range(0,len(words))}
26:     return wrank,normedcumsum
27:
28:
29:
30: with open('simple-20160801-1-article-per-line','rb') as file:
31:     content=file.readlines()
32: file.close()
33:
34:
35: zipf_probabilities = {' ': 0.17840450037213465, '1': 0.004478392057619917, '0': 0
36: uniform_probabilities = {' ': 0.1875, 'a': 0.03125, 'c': 0.03125, 'b': 0.03125, 'e
37:
38: # we transform the probabilities into commulative ones.
39: old_val=0.0
40: for key, value in zipf_probabilities.items():
41:     zipf_probabilities[key]=old_val+value
42:     old_val=zipf_probabilities[key]
```

```
43:
44: old_val=0.0
45: for key, value in uniform_probabilities.items():
46:     uniform_probabilities[key]=old_val+value
47:     old_val=uniform_probabilities[key]
48:
49: # we start the sampling here by looping on each doc and each letter in in each lin
50: # looping on N. The i generate 2 characters each using its own probability dist. T
51: # into the corresponding files every 1000 characters
52: print("Sampling started")
53: string=""
54: loop_index=0
55: zipf_probabilities_generated=[]
56: string=list(" "*1000)
57: string_index=0
58: string2=list(" "*1000)
59: while loop_index<len(content):
60:     loop_index_2=0
61:     line=content[loop_index].decode().lower()
62:     while loop_index_2<len(line):
63:         char=retreive_char(zipf_probabilities)
64:         char2=retreive_char(uniform_probabilities)
65:         string[string_index]=char
66:         string2[string_index]=char2
67:         string_index=string_index+1
68:         if(string_index==1000):
69:             zipf.write("".join(string))
70:             uniform.write("".join(string2))
71:             string_index=0
72:
73:         loop_index_2=loop_index_2+1
74:     loop_index=loop_index+1
75:     if(loop_index%5000==0):
76:         print("Sampling index:"+str(loop_index))
77: zipf.close()
78: uniform.close()
79: print("Done sampling")
80:
81: #we calculate CDF for simple english wiki, generated zipf and uniform dists
82:
83: with open('simple-20160801-1-article-per-line','r') as file:
84:     wrank,normedcumsum = getFileCDF(file)
85: file.close()
86: print("simple english wiki CDF calculation done")
87:
88: with open('zipf.txt','r') as zipf_file:
89:     zipf_wrank, zipf_normedcumsum = getFileCDF(zipf_file)
90: zipf_file.close()
91: print("zipf CDF calculation done")
```

```
92:
93:
94: with open('uniform.txt','r') as uniform_file:
95:     uniform_wrank, uniform_normedcumsum = getFileCDF(uniform_file)
96: uniform_file.close()
97: print("uniform CDF calculation done")
98:
99: #here i calculate the maximum point wise distance for both 2 generated corpuses
100: print("calculating the maximum point wise distance for zipf")
101: print("zipf max point= "+str(max(list(map(operator.sub, normedcumsum, zipf_normed
102: print("calculating the maximum point wise distance for uniform")
103: print("uniform max point= "+str(max(list(map(operator.sub, normedcumsum, uniform_
104:
105:
106: print("plotting now")
107:
108:
109: plt.title("Exercise 2")
110: plt.xlabel('Word Rrank')
111: plt.ylabel('CDF')
112: plt.loglog(list(range(len(wrank))), normedcumsum)
113: plt.loglog(list(range(len(zipf_wrank))), zipf_normedcumsum)
114: plt.loglog(list(range(len(uniform_wrank))), uniform_normedcumsum)
115: plt.show()
```

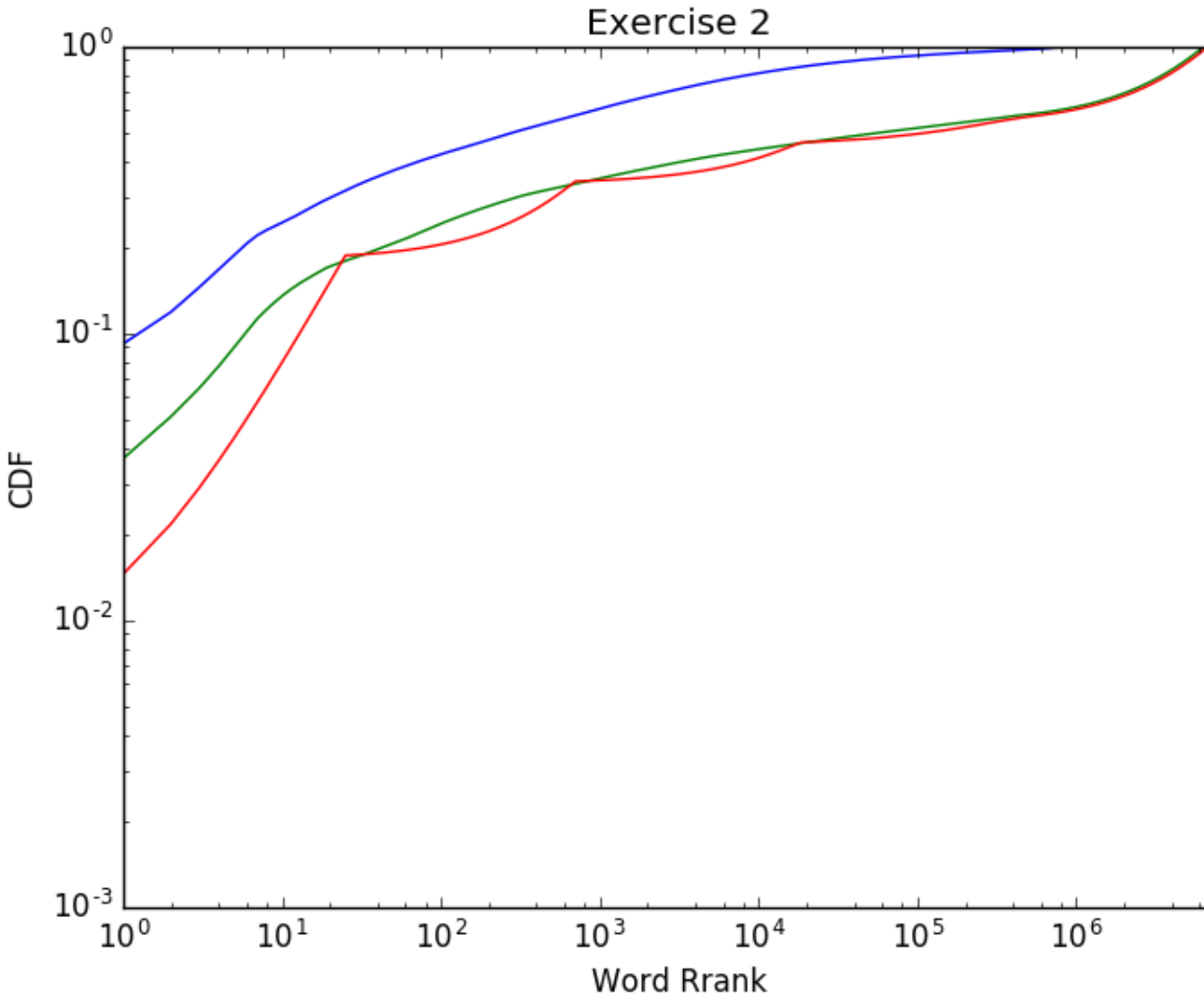
Exercise 2
CDF
Word Rrank
10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2
10^3
10^4
10^5
10^6

Figure 1

3 Understanding of the cumulative distribution function (10 points)

Write a fair 6-side die rolling simulator. A fair die is one for which each face appears with equal likelihood. Roll two dice simultaneously n (=100) times and record the sum of both dice each time.

1. Plot a readable histogram with frequencies of dice sum outcomes from the simulation.
2. Calculate and plot cumulative distribution function.
3. Answer the following questions using CDF plot:

   What is the median sum of two dice sides? Mark the point on the plot.

   What is the probability of dice sum to be equal or less than 9? Mark the point on the plot.
4. Repeat the simulation a second time and compute the maximum point-wise distance of both CDFs.
5. Now repeat the simulation (2 times) with n=1000 and compute the maximum point-wise distance of both CDFs.
6. What conclusion can you draw from increasing the number of steps in the simulation?

1. Figure 2

2. Figure 3

3.Sum count= '12': 9, '10': 2, '2': 3, '4': 4, '7': 6, '9': 5, '3': 7, '8': 8, '6': 1, '11': 11, '5': 10

star is where the median is, which is around the 6th ranked sum which is 7 circle is where the fifth ranked element is (9) with a prob close to 0.41.

4. Figure 4 the distance was 0.055.

5. It gave me a distance of 0.00999 which means it's better prediction the more we increase n.

6. In conclusion it gives a larger pool for randomness for the case giving much more accurate results. 100 trials are very less descriptive than 1000.

```
1: import random
2: import numpy as np
3: import matplotlib.pyplot as plt
4: import operator
5:
6: def roll( dict ):
7:     random_number=random.random()
8:     output=' '
9:     for key, value in dict.items():
```

```
10:         if(value>=random_number):
11:             output=key
12:             break
13:     return output
14:
15: def getFileCDF(dict):
16:         words=[]
17:         frequencies=[]
18:         for key, value in dict.items():
19:                 words.append(key)
20:                 frequencies.append(value)
21:         cumsum = np.cumsum(frequencies)
22:         normedcumsum = [x/float(cumsum[-1]) for x in cumsum]
23:         wrank = {words[i]:i+1 for i in range(0,len(words))}
24:         return wrank,normedcumsum
25: prob_dist = {'1': 1/6, '2': 1/6, '3': 1/6, '4': 1/6, '5': 1/6, '6': 1/6}
26:
27: old_val=0.0
28: old_val=0.0
29: for key, value in prob_dist.items():
30:     prob_dist[key]=old_val+value
31:     old_val=prob_dist[key]
32:
33: output_dict={'2':0,'3':0,'4':0,'5':0,'6':0,'7':0,'8':0,'9':0,'10':0,'11':0,'12':0
34:
35: loop_index=0
36: while loop_index<=100:
37:         n1=roll(prob_dist)
38:         n2=roll(prob_dist)
39:         sum_of=int(n1)+int(n2)
40:         sum_of=str(sum_of)
41:         output_dict[sum_of]=output_dict[sum_of]+1
42:         loop_index=loop_index+1
43:
44: a,cdf=getFileCDF(output_dict)
45:
46: print("sum count= ")
47: print(a)
48: l=[0]*11
49: for key, value in output_dict.items():
50:         l[int(key)-2]=value
51:
52: plt.title("Exercise 3")
53: plt.xlabel('Dice outcome')
54: plt.ylabel('Frequency')
55: plt.scatter([2,3,4,5,6,7,8,9,10,11,12],l)
56: plt.show()
57:
58: plt.title("Exercise 3")
```

```
plt.xlabel('Sum Rank')
plt.ylabel('CDF')
plt.plot([1,2,3,4,5,6,7,8,9,10,11],cdf)
plt.show()

output_dict={'2':0,'3':0,'4':0,'5':0,'6':0,'7':0,'8':0,'9':0,'10':0,'11':0,'12':0

loop_index=0
while loop_index<=100:
        n1=roll(prob_dist)
        n2=roll(prob_dist)
        sum_of=int(n1)+int(n2)
        sum_of=str(sum_of)
        output_dict[sum_of]=output_dict[sum_of]+1
        loop_index=loop_index+1

a2,cdf2=getFileCDF(output_dict)

plt.title("Exercise 3")
plt.xlabel('Sum Rank')
plt.ylabel('CDF')
plt.plot([1,2,3,4,5,6,7,8,9,10,11],cdf)
plt.plot([1,2,3,4,5,6,7,8,9,10,11],cdf2)
plt.show()
print("point wise distance= "+str(max(list(map(operator.sub, cdf, cdf2)))))#
```

3.1 Hints

1. You can use function from the lecture to calculate rank and normalized cumulative sum for CDF.
2. Do not forget to give proper names of CDF plot axes or maybe even change the ticks values of x-axis.

3.2 Only for nerds and board students (0 Points)

Assuming 20 groups of students. What is the likelihood that at least two groups come up with the same histograms in the case for n (=100)?

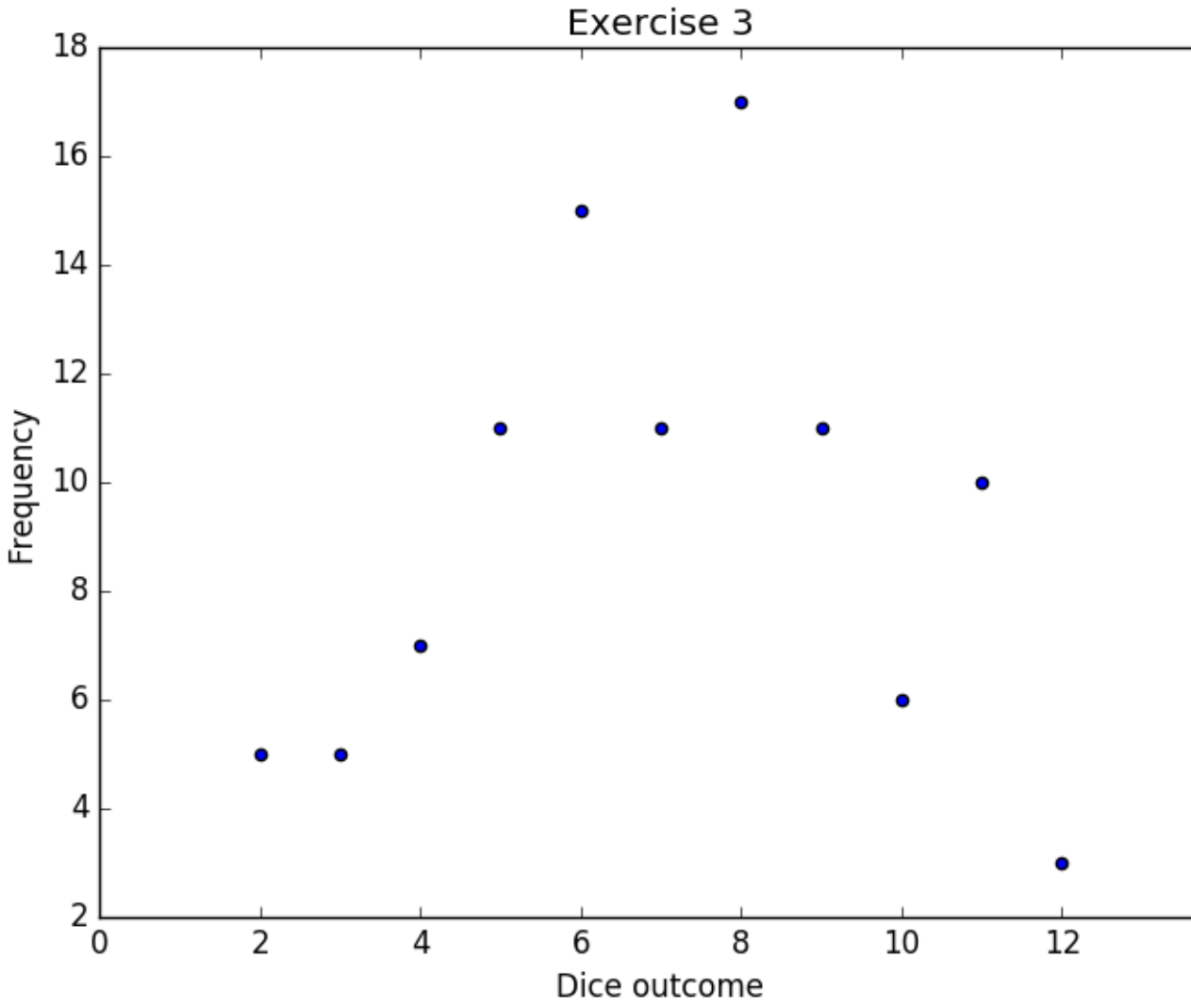
Exercise 3
Frequency
Dice outcome
18
16
14
12
10
8
6
4
2
0
2
4
6
8
10
12

Figure 2

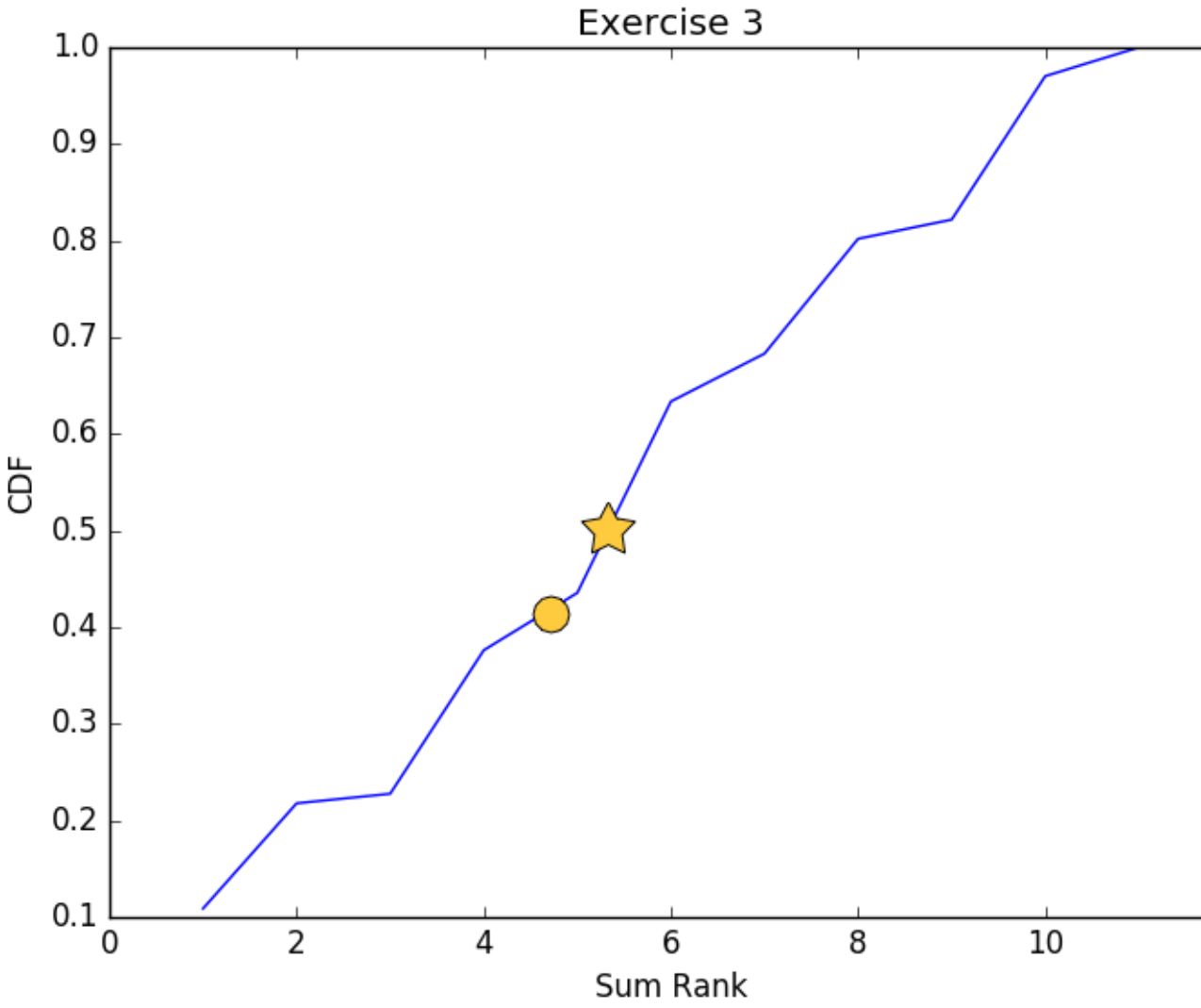

Figure 3

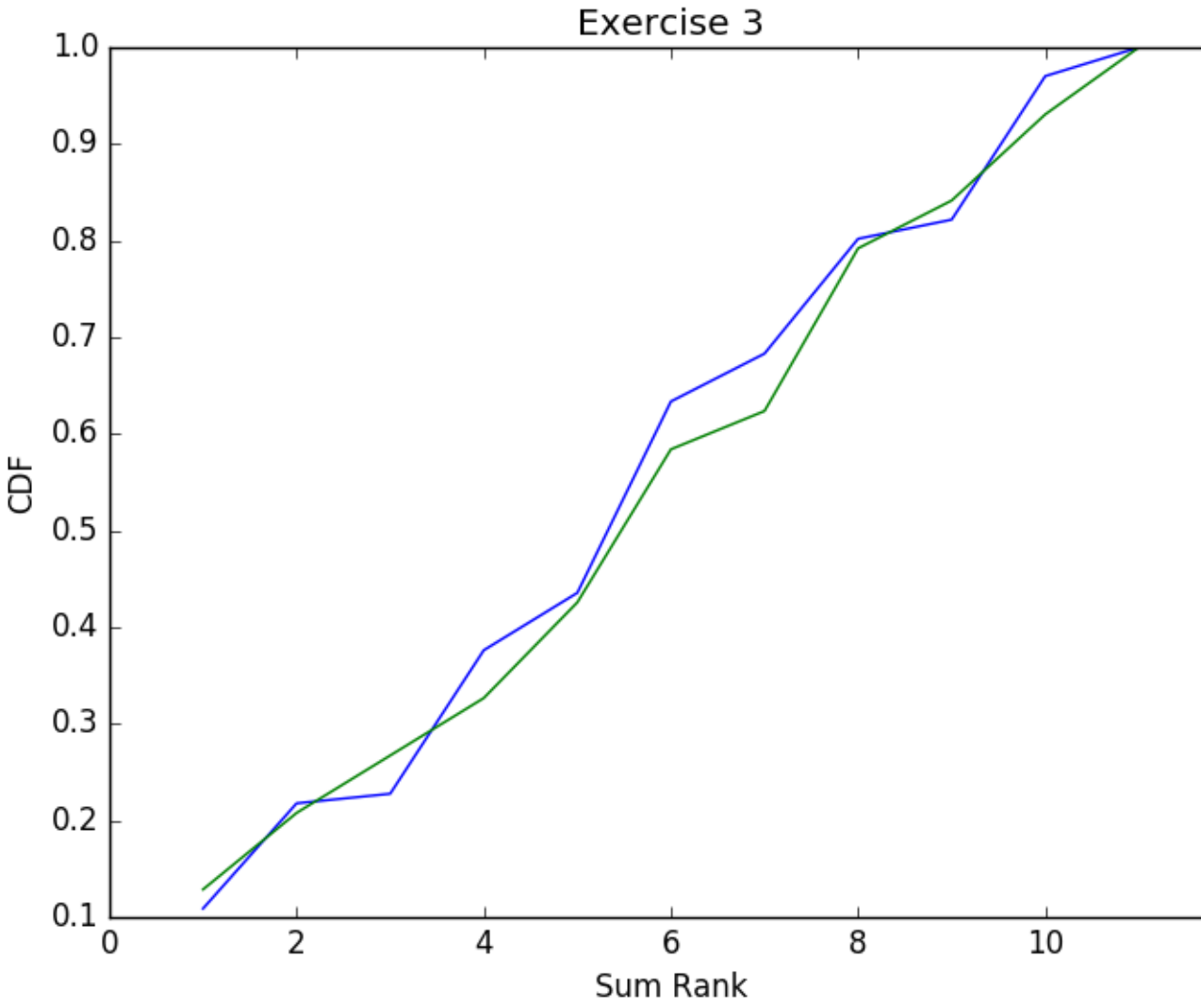

Figure 4

Important Notes

Submission

- Solutions have to be checked into the github repository. Use the directory name `groupname/assignment7/` 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`.