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.

Group Name : Yankee
Candidates :

1. Sabin Bhattarai - 216203590
   sbhattarai@uni-koblenz.de
2. Biplov K.C. - 216203865
   biplov@uni-koblenz.de
3. Syed Salman Ali. - 216203923
   salmanali@uni-koblenz.de

1 Modelling 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.

Answer 1.1

Semantic meaning of a document is given by the word and also the syntactic structure. We after observing the common pattern of usage of words in document 1 and document 2 (i.e. ”web science” occurs in both documents) come to conclusion that there is a meaning relation between two documents D1 and D2 and thus should be similar. Whereas D3 do not describe or state the word ”web science” and in our opinion has no common pattern to other documents.

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?
2. What does each dimension of the vector space stand for?
3. How many dimensions does the vector space have?
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.
6. Calculate the cosine similarity between all three pairs of vectors.

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

Answer 1.2

1.

Here,

Let V = $< \vec{this}$, $\vec{is}$, $\vec{a}$, $\vec{text}$, $\vec{about}$, $\vec{web}$, $\vec{science}$, $\vec{covering}$, $\vec{the}$, $\vec{analysis}$, $\vec{of}$, $\vec{corpora}$, $\vec{scientific}$, $\vec{methods}$, $\vec{are}$, $\vec{used}$, $\vec{to}$, $\vec{analyze}$, $\vec{webpages}$ $>$
be the vector spanned by the unique words shown as the vectors above.

i.e. ”this”, ”is”, ”a” , ”text”, ”about” , ”web” , ”science” , ”covering” , ”the” , ”analysis” , ”of” , ”corpora” , ”scientific” , ”methods” , ”are” ”used” , ”to” ,”analyze” , ”webpages” are unique words.

Thus,
We would need **19** base vectors to model the documents of this corpus.

2.

Each dimension of the vector space stands for unique words in our corpus. Also known as terms/token.

3.

Vector space has 19 dimensions.

4.

The following is the table that maps each words of the documents to the base vectors.

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th	13th	14th	15th	16th	17th	18th	19th
$\vec{this}$	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\vec{is}$	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\vec{a}$	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\vec{text}$	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\vec{about}$	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\vec{web}$	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$\vec{science}$	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
$\vec{covering}$	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
$\vec{the}$	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
$\vec{analysis}$	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
$\vec{of}$	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
$\vec{corpora}$	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
$\vec{scientific}$	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
$\vec{methods}$	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
$\vec{are}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
$\vec{used}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
$\vec{to}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
$\vec{analyze}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
$\vec{webpages}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 1: Mapping each word in document to base vector

5.

Firstly let us observe at the following frequency table which states the number of times a word appears in a document. i.e. The word "this" appears 1 times in D1 document and 0 on the others.

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

Table 2: Term frequency table.

Let us now say,
D1 has document vector $\vec{d1}$,
D2 has document vector $\vec{d2}$, and
D3 has document vector $\vec{d3}$

We know,

$$\vec{d_1} = \sum_{i=1}^{19} tf(w_i, D_1)\vec{w}_i$$

where,
tf(w_i, D_1) is the term frequency of the unique word in the document D1.

$\vec{w}_i$ is the base vector for the unique word.

Note: Since not all words occur in document 1, most of the term frequency calculation becomes 0. Thus, we omit representing them. Note we also refer to Table 2. for calculation

Referring to the term frequency table, we have,

$\vec{d_1}$ = tf('this', D_1) $\vec{this}$ + tf('is', D_1)$\vec{is}$ + tf('a', D_1)$\vec{a}$ + tf('text', D_1) $\vec{text}$ +tf('about', D_1) $\vec{about}$ + tf('web', D_1) $\vec{web}$ + tf('science', D_1)$\vec{science}$
or, $\vec{d_1}$ =

$$1*\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1*\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1*\begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1*\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1*\begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1*\begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1*\begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1*\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

i.e, $\vec{d_1}$ =

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Similarly,

$\vec{d_2}$ = tf('web', D_2) $\vec{web}$ + tf('science', D_2)$\vec{science}$ + tf('is', D_2)$\vec{is}$ + tf('covering', D_2)$\vec{covering}$ + tf('the', D_2) $\vec{the}$ +tf('analysis', D_2) $\vec{analysis}$ + tf('of', D_2) $\vec{of}$ + tf('text', D_2)$\vec{text}$ + tf('corpora', D_2)$\vec{corpora}$

i.e., $\vec{d}_2 =$

$$1 * \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1 * \begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1 * \begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1 * \begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1 * \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1 * \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1 * \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1 * \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1 * \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1 * \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

i.e., $\vec{d}_2 =$

$$\begin{bmatrix}0\\1\\0\\1\\0\\1\\1\\1\\1\\1\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

Again,

$\vec{d_3}$ = tf('scientific', D_3) $\vec{scientific}$ + tf('methods', D_3)$\vec{methods}$ + tf('are', D_3)$\vec{are}$ + tf('used', D_3) $\vec{used}$ +tf('to', D_3) $\vec{to}$ + tf('analyze', D_3) $\vec{analyze}$ + tf('webpages', D_3)$\vec{webpages}$

i.e. $\vec{d_3}$ =

$$1*\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}+1*\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}+1*\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}+1*\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}+1*\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix}+1*\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix}+1*\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}+1*\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$$

i.e. $\vec{d_3} =$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

6.

We know,

Three pairs of vectors would be as follows:
$(\vec{d_1}, \vec{d_2})$, $(\vec{d_2}, \vec{d_3})$, $(\vec{d_1}, \vec{d_3})$

Firstly, let us find cosine for vector pair $(\vec{d_1}, \vec{d_2})$, which is given by:

$$cos(\theta) = \frac{< \vec{d_1}.\vec{d_2} >}{||\vec{d_1}||.||\vec{d_2}||}$$

where, $< \vec{d_1}.\vec{d_2} >$ is the scalar product obtained as,

$$\sum_{k=1}^{n} (\vec{d_1})_k (\vec{d_2})_k$$

i.e.

$$\sum_{k=1}^{n} tf(w_k, D_1), tf(w_k, D_2)$$

Since, most term frequency is 0, we obtain following
$< \vec{d_1}, \vec{d_2} > = 1 + 1 + 1 + 1 = 4$

Also,
$||\vec{d_1}|| = \sqrt{< d_1, d_1 >}$
i.e. $||\vec{d_1}|| = \sqrt{1+1+1+1+1+1+1}$
i.e. $||\vec{d_1}|| = \sqrt{7}$

$||\vec{d_2}|| = \sqrt{< d_2, d_2 >}$
i.e. $||\vec{d_2}|| = \sqrt{1+1+1+1+1+1+1+1+1}$
i.e. $||\vec{d_2}|| = \sqrt{9}$

Therefore,

$$cos(\theta) = \frac{4}{\sqrt{7}.\sqrt{9}}$$
$$cos(\theta) = 0.504$$

Now Let us evaluate for vector pair $(\vec{d_1}, \vec{d_3})$

$$cos(\theta) = \frac{< \vec{d_1}.\vec{d_3} >}{||\vec{d_1}||.||\vec{d_3}||}$$

Where, $< \vec{d_1}.\vec{d_3} >$ is the scalar product obtained as,

$$\sum_{k=1}^{n} (\vec{d_1})_k (\vec{d_3})_k$$

i.e.

$$\sum_{k=1}^{n} tf(w_k, D_1), tf(w_k, D_3)$$

Since, most term frequency is 0, we obtain following
$< \vec{d_1}, \vec{d_3} > = 0$
Since, $< \vec{d_1}, \vec{d_3} > = 0$

$$cos(\theta) = \frac{0}{||\vec{d_1}||.||\vec{d_3}||} = 0$$

Again, for vector pair $(\vec{d_2}, \vec{d_3})$

$$cos(\theta) = \frac{< \vec{d_2}.\vec{d_3} >}{||\vec{d_1}||.||\vec{d_3}||}$$

where,
$< \vec{d_2}.\vec{d_3} >$ is the scalar product obtained as,

$$\sum_{k=1}^{n} (\vec{d_2})_k (\vec{d_3})_k$$

i.e.

$$\sum_{k=1}^{n} tf(w_k, D_2), tf(w_k, D_3)$$

Since, most term frequency is 0, we obtain following
$< \vec{d_2}, \vec{d_3} > = 0$

Since, $< \vec{d_2}, \vec{d_3} > = 0$

$$cos(\theta) = \frac{0}{||\vec{d_2}||.||\vec{d_3}||} = 0$$

7.
According to the cosine similarity we observe the greater similarity measure between Document 1 and Document 2 since it had 0.504 as similarity measure compared to 0 for other pairs.

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.

Answer 1.3

Yes the model matched our expectations from the first subtask. As we stated in first subtask the occurrences of certain words in both set of documents meant the semantics would match. Now here in the vector space when we evaluate the cosine theta for similarity measure, the frequency of the words occurring in both set of documents influenced the value of cosine theta. This meant the more the words matched, we would have larger cosine theta value. The higher the value the better similarity can be observed.Thus Document 1 and Document 2 were similar as per our observation in subtask 1.

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.

```
# Code for Question 2.1

import collections
import numpy as np
import ast
import bisect
import random
import matplotlib.pyplot as plt
from threading import Thread

counterAllCharInEnglish = 0
THREADS = 30

#Parses the file. Helper function for countCharactersSpaces()
def getEachLinesFromFile(filename):
    with open(filename , encoding="utf8") as fp:
        try:
            content = fp.read()
        except:
            content = ""
    return content

def writeTextToFile(filename , text):
    with open(filename , 'w+' , encoding="utf8") as fp:
        try:
            content = fp.write(text)
        except:
            content = ""
    return content

```

```
31:
32: # Get the total characters in Simple English file
33: def getCharactersInSimpleEnglish(filename):
34:     getSimpleEnglishText = getEachLinesFromFile(simpleEnglishFilename)
35:     return len(getSimpleEnglishText)
36:
37:
38: #Returns probabilities in the file as dictionaries. Each dictionaries are
39: #elements of list. Note we here only have Zipf and uniform probabilities.
40: def readProbabilitiesFromFileToDict(filename):
41:     with open(filename, 'r') as f:
42:         s = f.read()
43:
44:     # ------------------------ First part for Zipf   --------------------------
45:     if s.find('{') < 0:
46:         return None
47:     s = s[s.find('{'):]
48:
49:     if s.find('}') < 0:
50:         return None
51:     zipf_probabilities = s[:s.find('}') + 1]
52:
53:     # -------------------------Second part for Uniform--------------------------
54:     # Getting the remaining probabilities after deducing Zipf
55:     remainingProbabilities = s[s.find('}') + 1:]
56:
57:     # Now we strip the remaningProb to get uniform probabilities
58:     if remainingProbabilities.find('{') < 0:
59:         return None
60:     remainingProbabilities = remainingProbabilities\
61:                                              [remainingProbabilities.find('{'):]
62:
63:     if remainingProbabilities.find('}') < 0:
64:         return None
65:     uniformProbabilities = remainingProbabilities\
66:                                           [:remainingProbabilities.find('}') + 1]
67:
68:     zipf_probabilitiesDict = ast.literal_eval(zipf_probabilities)
69:     uniformProbabilitiesDict = ast.literal_eval(uniformProbabilities)
70:     return [zipf_probabilitiesDict , uniformProbabilitiesDict]
71:
72: #-------------------------------------------------------------------------------
73: # Section RandomTextGenerate STARTS
74: #-------------------------------------------------------------------------------
75:
76: # This section is for generating random text when the characters to be used are
77: # sent with its associated CDF
78: # Note we create 30 threads
79:
```

```
80: def threadToGetRandomText(allKeys , associatedCDF , result , indexer):
81:     characters = []
82:     for x in range(0, counterAllCharInEnglish // THREADS):
83:         randomValue = random.random()
84:         index = bisect.bisect(associatedCDF, randomValue)
85:         characters.append(allKeys[index])
86:     text = "".join(characters)
87:     result[indexer] = text
88:     return text
89:
90: def getRandomTextAsPerProbabilityDist(allKeys , associatedCDF):
91:     threads = [None] * THREADS
92:     results = {}
93:     i = 0;
94:     for i in range(0, len(threads)):
95:         threads[i] = Thread(target=threadToGetRandomText, \
96:                                     args=(allKeys , associatedCDF, results, i))
97:         threads[i].start()
98:
99:     for i in range(len(threads)):
100:         threads[i].join()
101:
102:     return ''.join(list(results.values()))
103:
104: #-------------------------------------------------------------------------
105: # Section RandomTextGenerate ENDS
106: #-------------------------------------------------------------------------
107:
108:
109: def getWordWithItsProbabilities(text):
110:     frequencyEachWord_S = collections.Counter(text.split()).most_common()
111:     #Here we get probabilities for each character
112:     sumOfTotalCharOccurrence = sum([frequency for (key,frequency) in \
113:                                                     frequencyEachWord_S])
114:     probabilityForEachWordDict = {}
115:     for (key , frequency) in frequencyEachWord_S:
116:         probabilityForEachWordDict[key] = frequency / \
117:                                                     sumOfTotalCharOccurrence
118:     print ("totalSum Word" ,sumOfTotalCharOccurrence )
119:     return probabilityForEachWordDict
120:
121: def cdfCalculation(probDictForCDFCalc):
122:     arrayForCDFCalc = list(probDictForCDFCalc.values())
123:     a = np.array(arrayForCDFCalc) #Gets us CDF
124:     cdfEvalDict = np.cumsum(a)
125:     return (list(probDictForCDFCalc.keys()) , cdfEvalDict)
126:
127:
128: def performKolmogorovSmirnovTest(cdfsMain , cdfsToCompare):
```

```
129:      maxPointwiseDistance = max([abs(cdf_S - cdf_Zipf) \
130:                                  for (cdf_S , cdf_Zipf) in \
131:                                                zip(cdfsMain , cdfsToCompare)])
132:      return maxPointwiseDistance
133:
134: #---------------------------------------------------------------------------
135: # Section Plots STARTS
136: #---------------------------------------------------------------------------
137:
138: # This section is for plotting rank frequency diagram
139:
140: def drawPlot(listElements_S , listElements_Z , listElements_U , xLabel , yLabel):
141:     x_S = [x for x in range(1, len(listElements_S)+1)]
142:     y_S = np.array(listElements_S)
143:     x_Z = [x for x in range(1, len(listElements_Z)+1)]
144:     y_Z = np.array(listElements_Z)
145:     x_U = [x for x in range(1, len(listElements_U)+1)]
146:     y_U = np.array(listElements_U)
147:     plt.figure(figsize=(12,9))
148:     plt.plot(x_S, y_S , 'r', label = "Simple English")
149:     plt.plot(x_Z, y_Z , 'b', label = "Zips Distribution Words")
150:     plt.plot(x_U, y_U , 'g', label = "Uniform Distribution Words")
151:
152:     plt.xlabel(xLabel)
153:     plt.ylabel(yLabel)
154:     plt.yscale('log')
155:     plt.xscale('log')
156:     plt.legend(loc='upper right')
157:     plt.grid()
158:     plt.show()
159:
160: def drawCDFPlot(listElements_S , listElements_Z , listElements_U , xLabel , yLabel
161:     x_S = [x for x in range(1, len(listElements_S)+1)]
162:     y_S = np.array(listElements_S)
163:     x_Z = [x for x in range(1, len(listElements_Z)+1)]
164:     y_Z = np.array(listElements_Z)
165:     x_U = [x for x in range(1, len(listElements_U)+1)]
166:     y_U = np.array(listElements_U)
167:     plt.figure(figsize=(12,9))
168:     plt.plot(x_S, y_S , 'r', label = "Simple English")
169:     plt.plot(x_Z, y_Z , 'b' , label = "Zips Distribution Words")
170:     plt.plot(x_U, y_U , 'g' , label = "Uniform Distribution Words")
171:
172:     plt.xlabel(xLabel)
173:     plt.ylabel(yLabel)
174:     plt.xscale('log')
175:     plt.yscale('log')
176:     plt.ylim(0, 1.2)
177:     plt.legend(loc='upper right')
```

```
178:     plt.grid()
179:     plt.show()
180:
181: #-------------------------------------------------------------------------------
182: # Section Plots ENDS
183: #-------------------------------------------------------------------------------
184:
185:
186:
187: def mainFunction(simpleEnglishFilename , probabilitiesFilename):
188:     # This one is for simple english wikipedia.
189:     # Here we get the probability for each character in a dictionary format.
190:     # Then we get CDF for each words in tuple format
191:     getSimpleEnglishText = getEachLinesFromFile(simpleEnglishFilename)
192:     probabilityForEachWordDict = getWordWithItsProbabilities(getSimpleEnglishText)
193:     (allKeys_S, associatedCDF_S ) = cdfCalculation(probabilityForEachWordDict)
194:
195:
196:     # This one is for probabilistic distribution given in file.
197:     # i.e. probability distribution and uniform distribution
198:     probabilityDistFromFile =  readProbabilitiesFromFileToDict(\
199:                                                     probabilitiesFilename)
200:     (allKeys_Zipf, associatedCDF_Zipf ) = cdfCalculation(\
201:                                                   probabilityDistFromFile[0])
202:     (allKeys_Unif, associatedCDF_Unif ) = cdfCalculation(\
203:                                                   probabilityDistFromFile[1])
204:
205:     #following gives the text for the Zipf and uniform distribution
206:     text_Zipf = getRandomTextAsPerProbabilityDist(allKeys_Zipf , \
207:                                                  associatedCDF_Zipf)
208:     text_Unif = getRandomTextAsPerProbabilityDist(allKeys_Unif , \
209:                                                  associatedCDF_Unif)
210:
211:
212:     probabilityForEachWordDict_Zipf = getWordWithItsProbabilities(text_Zipf)
213:     (allKeys_NewZipf, associatedCDF_NewZipf ) = cdfCalculation(\
214:                                             probabilityForEachWordDict_Zipf)
215:
216:     probabilityForEachWordDict_Unif = getWordWithItsProbabilities(text_Unif)
217:     (allKeys_NewUnif, associatedCDF_NewUnif ) = cdfCalculation(\
218:                                             probabilityForEachWordDict_Unif)
219:
220:
221:     #To put both obtained text into a file
222:     #print('Zipf text' ,text_Zipf )
223:     #print('\nUnif text' ,text_Zipf)
224:
225:     writeTextToFile("ZipfCreatedText.txt",text_Zipf)
226:     writeTextToFile("UnifCreatedText.txt",text_Unif)
```

```
227:
228:     #-------------------------------------------------------------------------
229:     # Plots for Rank Frequency STARTS
230:     #-------------------------------------------------------------------------
231:     #Note: _S is for simple English
232:     frequencyEachWord_S = collections.Counter(getSimpleEnglishText.split())\
233:                                                                 .most_common()
234:     listForRankFrequencyDiag_S = [frequency for (word , frequency) \
235:                                                 in frequencyEachWord_S]
236:
237:     #Note _Z is for Zipf
238:     frequencyEachWord_Z = collections.Counter(text_Zipf.split()).most_common()
239:     listForRankFrequencyDiag_Z = [frequency for (word , frequency) \
240:                                                 in frequencyEachWord_Z]
241:
242:     #Note_U is for Uniform
243:     frequencyEachWord_U = collections.Counter(text_Unif.split()).most_common()
244:     listForRankFrequencyDiag_U = [frequency for (word , frequency) \
245:                                                 in frequencyEachWord_U]
246:
247:     drawPlot(listForRankFrequencyDiag_S , listForRankFrequencyDiag_Z, \
248:              listForRankFrequencyDiag_U, "x = Rank" , "y = Number of occurences")
249:     #-------------------------------------------------------------------------
250:     # Plots for Rank Frequency ENDS
251:     #-------------------------------------------------------------------------
252:
253:     #-------------------------------------------------------------------------
254:     # Plots for Rank and CDF STARTS
255:     #-------------------------------------------------------------------------
256:
257:     drawCDFPlot(associatedCDF_S , associatedCDF_NewZipf, associatedCDF_NewUnif,\
258:                                         "x = Rank" , "y = Cumulative Frequency")
259:
260:     #-------------------------------------------------------------------------
261:     # Plots for Rank and CDF ENDS
262:     #-------------------------------------------------------------------------
263:
264:     # Now we perform Kolmogorov Smirnov test  by calculating the maximum
265:     # pointwise distance of CDFs
266:     maxPointWiseD_S_Zipf = performKolmogorovSmirnovTest(associatedCDF_S , \
267:                                                         associatedCDF_NewZipf)
268:     maxPointWiseD_S_Unif = performKolmogorovSmirnovTest(associatedCDF_S , \
269:                                                         associatedCDF_NewUnif)
270:     print ("Obtained Maximum Pointwise Distance between simple English \
271:                                         and Zipf is : " , maxPointWiseD_S_Zipf)
272:     print ("Obtained Maximum Pointwise Distance between simple English \
273:                                         and Unif is : " , maxPointWiseD_S_Unif)
274:
275: if __name__ == "__main__":
```

```
276:     simpleEnglishFilename = "simple-20160801-1-article-per-line"
277:     probabilitiesFilename = "probabilities.py.txt"
278:     # Here we calculate the total characters in Simple English and
279:     # assign global variable
280:     counterAllCharInEnglish = getCharactersInSimpleEnglish(simpleEnglishFilename)
281:     print ("TotalChracters: " ,counterAllCharInEnglish)
282:     mainFunction(simpleEnglishFilename ,probabilitiesFilename )
```

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.

Answer 2.2

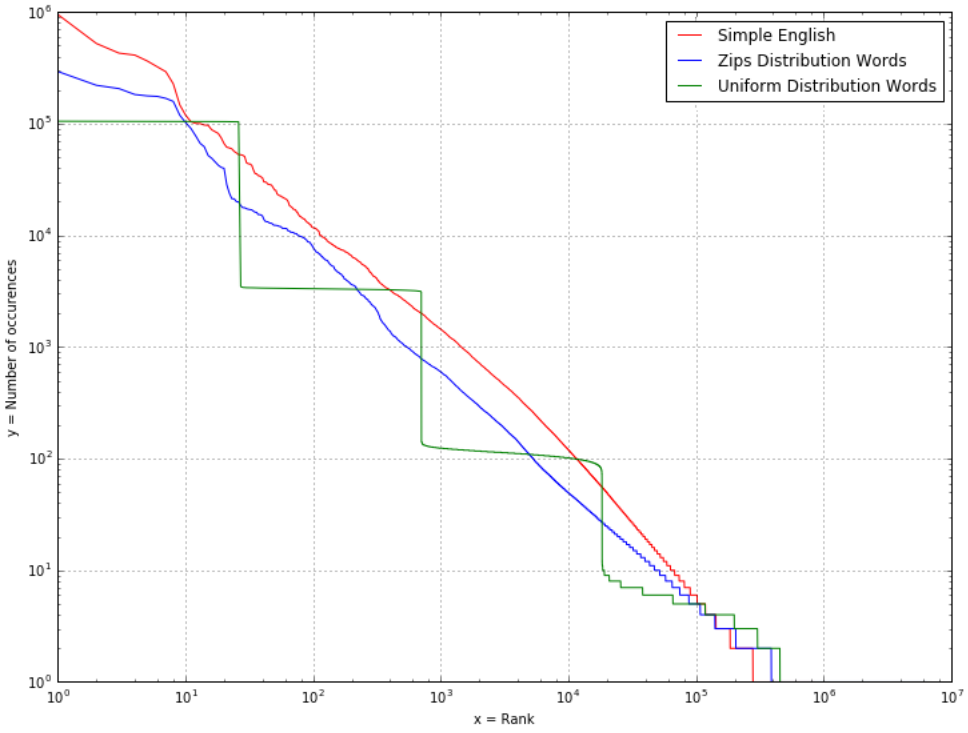

Figure 1: Word frequencies depending on word rank

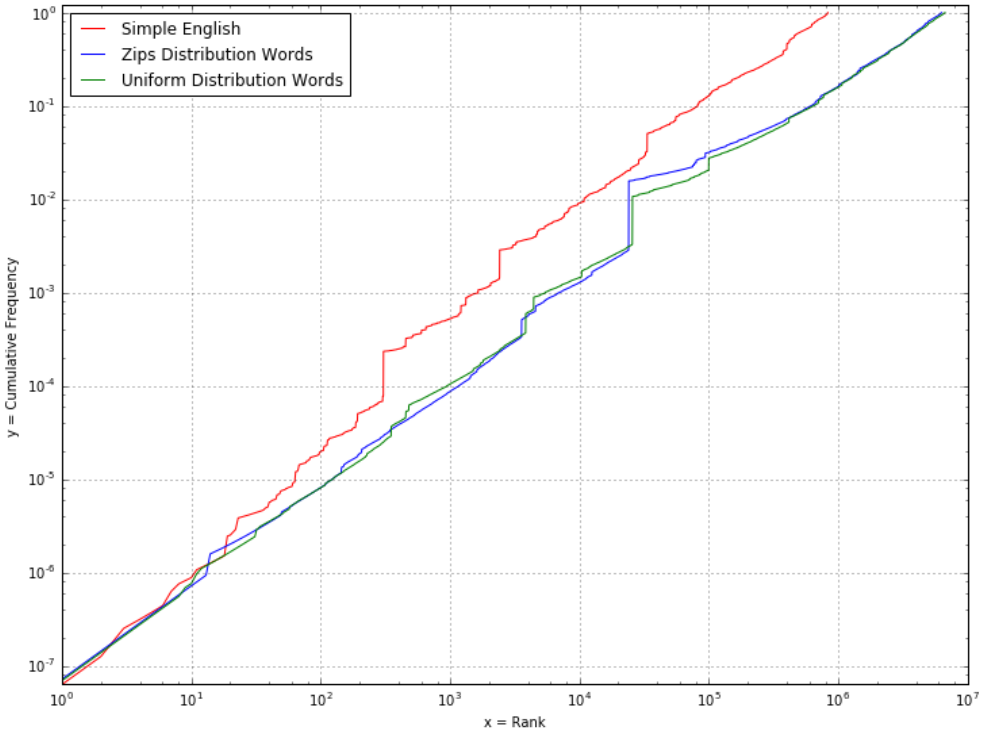

Figure 2: cumulative word probabilities depending on word rank

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?

Answer 2.3

Obtained Maximum Pointwise Distance between simple English and Zipf is : 0.860301461905
Obtained Maximum Pointwise Distance between simple English and Unif is : 0.861857581856

Although our model is far off, from Figure 2 and its resulting Kolmogorov Smirnov test We would choose data set created by Zipf distribution because it has lower distance measure with our original data set.This suggests Zipf distribution model reflects more to our descriptive model than data set created from uniform distribution

Two text corpora for second time

Obtained Maximum Pointwise Distance between simple English and Zipf is : 0.863258314669
Obtained Maximum Pointwise Distance between simple English and Unif is : 0.852805206287

Two text corpora for third time

Obtained Maximum Pointwise Distance between simple English and Zipf is : 0.857980183468
Obtained Maximum Pointwise Distance between simple English and Unif is : 0.862024209875

Our choice of a model earlier can be called off and be said inconsistent as we now observe that during the repetition of generating two text corpora our Pointwise distance differed in each iteration.

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.

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?

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.

Answer 3.1

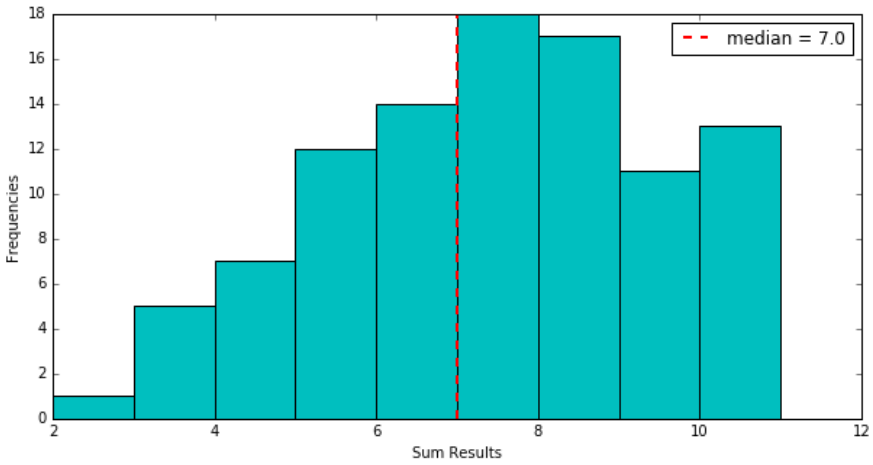

Figure 3: Histogram plot with frequencies of dice sum outcomes

---

Answer 3.2

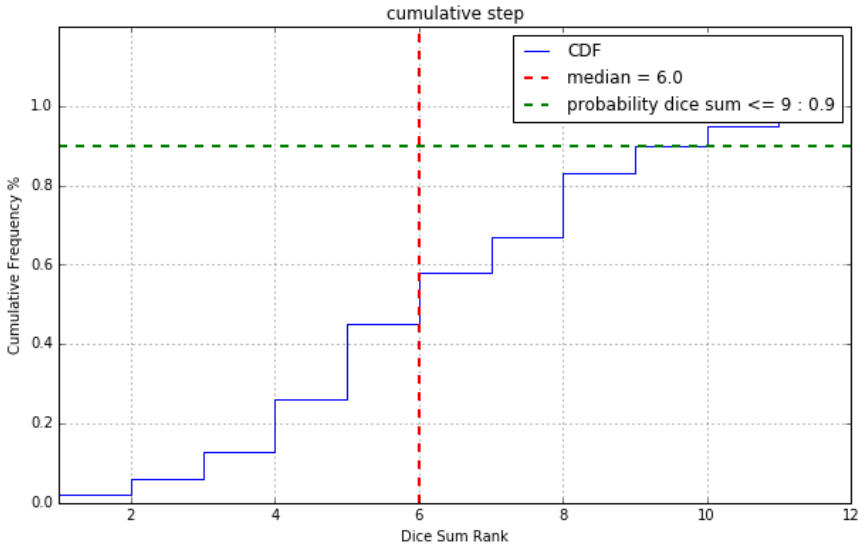

Figure 4: cumulative distribution function.

Answer 3.3

The median sum of two dice sides is 6.0. The line drawn with red colour in the diagram above represents the median.

The probability of dice sum to be equal or less than 9 is 0.9. The line drawn with green colour in the diagram above represents the probability.

Answer 3.4

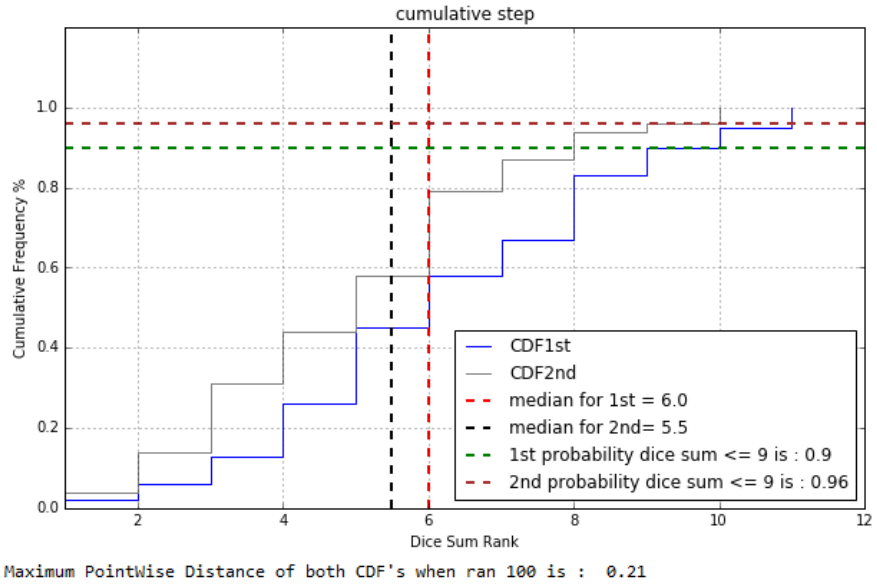

Figure 5: cumulative distribution function for two iterations.

As shown in the figure above, the maximum point-wise distance of both CDFs is **0.21**

Answer 3.5

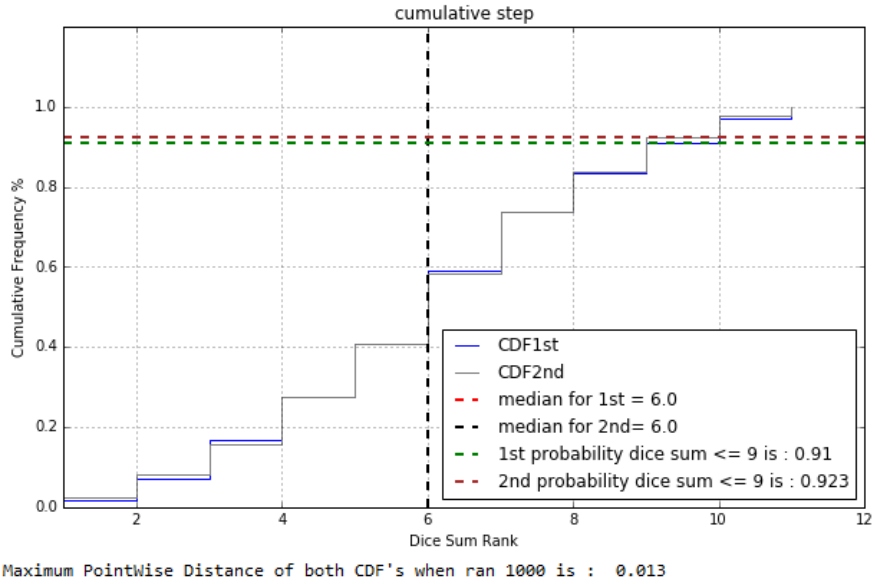

Figure 6: cumulative distribution function for two iterations.

Repeating the simulation (2 times) with n=1000 and computing the maximum point-wise distance of both CDFs gives us a value **0.013** as shown in figure above.

Answer 3.6

After increasing the number of steps in the simulation from 100 to 1000 we observed the point-wise distance of both CDFs (100) was higher compared to that of both CDFs (1000). i.e. $0.21 < 0.013$. This could mean that if we were to increase the number of dice roll to very large number, we would almost get the similar CDFs plot stating the best similarity in the model.

```
# Code for Question 3

import random
import numpy as np
import matplotlib.pyplot as plt
import collections

ROLLDICESIDES = 6
# roll two dice and get sum
def rollDiceGetSum():
    return random.randint(1, ROLLDICESIDES) + random.randint(1, ROLLDICESIDES)

def rollDiceAndGetListOfSums(n):
    return np.sort([rollDiceGetSum() for i in range(n)])

# draw a CDF plot for a given list of normalised data.
def drawCDFPlot(listElements , xLabel , yLabel):
    x_1 = [x for x in range(1, len(listElements)+1)]
    y_1 = np.array(listElements)

    plt.figure(figsize=(10,6))
    plt.plot(x_1, y_1 , drawstyle='steps-post' , color = 'b' , label = 'CDF')

    plt.xlabel(xLabel)
    plt.ylabel(yLabel)
    plt.ylim(0, 1.2)
    plt.title('cumulative step')

    plt.xlim(xmin=1)
    plt.axvline(np.median(x_1), color='r', linestyle='dashed', linewidth=2 , \
                                    label = "median = " + str(np.median(x_1)))

    print(x_1)
    plt.axhline(y_1[8], color='g', linestyle='dashed', linewidth=2 , \
                         label = "probability dice sum <= 9 : " + str(y_1[8]))

    plt.legend(loc='upper right')
    plt.grid()
    plt.show()

def drawCDFPlotTwo(listElements1 , listElements2 , xLabel , yLabel):
    x_1 = [x for x in range(1, len(listElements1)+1)]
    y_1 = np.array(listElements1)

    x_2 = [x for x in range(1, len(listElements2)+1)]
    y_2 = np.array(listElements2)

    plt.figure(figsize=(10,6))
    plt.plot(x_1, y_1 , drawstyle='steps-post' , color = 'b' , label = 'CDF1st')
```

```
50:     plt.plot(x_2, y_2 , drawstyle='steps-post' , \
51:                                                 color = 'gray' , label = 'CDF2nd')
52:
53:     plt.xlabel(xLabel)
54:     plt.ylabel(yLabel)
55:     plt.ylim(0, 1.2)
56:     plt.title('cumulative step')
57:     plt.xlim(xmin=1)
58:
59:     plt.axvline(np.median(x_1), color='r', linestyle='dashed', linewidth=2 , \
60:                               label = "median for 1st = " + str(np.median(x_1)))
61:
62:     plt.axvline(np.median(x_2), color='black', linestyle='dashed', linewidth=2,\
63:                                label = "median for 2nd= " + str(np.median(x_2)))
64:
65:     plt.axhline(y_1[8], color='g', linestyle='dashed', linewidth=2 , \
66:                     label = "1st probability dice sum <= 9 is : " + str(y_1[8]))
67:     plt.axhline(y_2[8], color='brown', linestyle='dashed', linewidth=2 , \
68:                     label = "2nd probability dice sum <= 9 is : " + str(y_2[8]))
69:
70:     plt.legend(loc='bottom right')
71:     plt.grid()
72:     plt.show()
73:
74: def drawHistogram(listElements , xLabel , yLabel, interval):
75:     print (listElements)
76:     x_1 = np.array(listElements)
77:     plt.figure(figsize=(10,5))
78:     plt.hist(x_1 , bins= range(0, 12, interval) , color='c' )
79:     plt.xlabel(xLabel)
80:     plt.ylabel(yLabel)
81:     plt.xlim(xmin=2)
82:     plt.axvline(np.median(x_1), color='r', linestyle='dashed', linewidth=2 , \
83:                                       label = "median = " + str(np.median(x_1)))
84:
85:     plt.legend(loc='upper right')
86:     plt.show()
87:
88:
89: def calculateCDF(result):
90:     frequencyNum = collections.Counter(result).most_common()
91:     total = sum([frequency for (key, frequency) in frequencyNum])
92:     probabilityForEachNumDict = {}
93:     for (key, frequency) in frequencyNum:
94:         probabilityForEachNumDict[key] = frequency / \
95:                                             total
96:
97:     arrayForCDFCalc = list(probabilityForEachNumDict.values())
98:     a = np.array(arrayForCDFCalc)
```

```
 99:     cdfEvalDict = np.cumsum(a)
100:     return (list(probabilityForEachNumDict.keys()), cdfEvalDict)
101:
102: def performKolmogorovSmirnovTest(cdfsMain , cdfsToCompare):
103:      maxPointwiseDistance = max([abs(cdf_S - cdf_Zipf) \
104:                                  for (cdf_S , cdf_Zipf) in \
105:                                                  zip(cdfsMain , cdfsToCompare)])
106:      return maxPointwiseDistance
107:
108:
109: if __name__ == "__main__":
110:     drawHistogram(rollDiceAndGetListOfSums(100) ,"Sum Results" ,"Frequencies", 1)
111:     (eachNumbers , associatedCDFs) = calculateCDF(rollDiceAndGetListOfSums(100))
112:     (eachNumbers , secondAssociatedCDFs) = calculateCDF(\
113:                                                  rollDiceAndGetListOfSums(100))
114:     drawCDFPlot(associatedCDFs , "Dice Sum Rank", "Cumulative Frequency %" )
115:     drawCDFPlotTwo(associatedCDFs , secondAssociatedCDFs, "Dice Sum Rank" , \
116:                                                      "Cumulative Frequency %")
117:     #Computing the maximum pointwise distance for both CDF's
118:     print ("Maximum PointWise Distance of both CDF's when ran 100 is : " , \
119:            performKolmogorovSmirnovTest(associatedCDFs , secondAssociatedCDFs))
120:
121:
122:     (eachNumbers1000 , associatedCDFs1000) = calculateCDF(\
123:                                                  rollDiceAndGetListOfSums(1000))
124:     (eachNumbers1000 , secondAssociatedCDFs1000) = \
125:                                    calculateCDF(rollDiceAndGetListOfSums(1000))
126:     drawCDFPlotTwo(associatedCDFs1000 , secondAssociatedCDFs1000, \
127:                                    "Dice Sum Rank" , "Cumulative Frequency %")
128:     #Computing the maximum pointwise distance for both CDF's
129:     print ("Maximum PointWise Distance of both CDF's when ran 1000 is : ",\
130:       performKolmogorovSmirnovTest(associatedCDFs1000 , secondAssociatedCDFs1000))
```

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)?

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`.