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: Echo

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.

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.

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:
1.1 Get a feeling for similarity as a human

D1 and D2 are most similar set of two documents to each other between these three documents, because they have more common words in their documents than any other two documents.

1.2 Model the documents as vectors and use the cosine similarity

- How many base vectors would be needed to model the documents of this corpus? answer: 19 base vectors, for each unique word in the corpus
- What does each dimension of the vector space stand for? answer: Each dimension represents each word in the corpus. The document is drawn in regards of the term frequency of that word in its document. For example, a dimension of w1 is there and the document vector (for example D1) uses the term frequency of w1 in its document which equals to 1.
- How many dimensions does the vector space have? answer: 19 dimensions
- Create a table to map words of the documents to the base vectors. answer: words vectors

(	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	) vector(w1)
(	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	) vector(w2)
(	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	) vector(w3)
(	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	) vector(w4)
(	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	) vector(w5)
(	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	) vector(w6)
(	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	) vector(w7)
(	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	) vector(w8)
(	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	) vector(w9)
(	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	) vector(w10)
(	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	) vector(w11)
(	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	) vector(w12)
(	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	) vector(w13)
(	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	) vector(w14)
(	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	) vector(w15)
(	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	) vector(w16)
(	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	) vector(w17)
(	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	) vector(w18)
(	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	) vector(w19)

- 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. answer:

D1 = w1,w2,w3,w4,w5,w6,w7
D2 = w6,w7,w2,w8,w9,w10,w11,w4,w12
D3 = w13,w14,w15,w16,w17,w18,w19
using term frequency model

		w19	w18	w17	w16	w15	w14	w13	w12	w11	w10	w9	w8	w7	w6	w5	w4	w3	w2	w1	
vector(D1)	(	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	)
vector(D2)	(	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	1	0	1	0	)
vector(D3)	(	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	)

- Calculate the cosine similarity between all three pairs of vectors. answer: using term frequency model:

  length(vector(D1)) = sqrt((1^2)*7) = 2.64
  length(vector(D2)) = sqrt((1^2)*9) = 3
  length(vector(D3)) = sqrt((1^2)*7) = 2.64

  similarity(vector(D1), vector(D2)) = (4)/(2.64*3) = 0.50
  similarity(vector(D2), vector(D3)) = (0)/(3*2.64) = 0
  similarity(vector(D1), vector(D3)) = (0)/(2.64*2.64) = 0

- According to the cosine similarity which 2 documents are most similar according to the constructed model. answer: (D1 and D2) are the most similar set of two documents according to the cosine similarity. Other sets of two documents have the similarity of (0)

1.3 Discussion

The results matched our expectations.

In the calculation of the cosine similarity, if two documents have a word in common then the value of the dot product of the two documents is calculated by multiplying the word frequency of that word in the first document by the word frequency of the same word in the second product (and that goes for all words in the document) other than that it'll be a zero field. Therefore, the documents that don't have words in common will end up having a zero dot product. Which will lead to a Zero similarity, when we divide it with the length of the vector. So basically, because document one and document two have four words in common we found a similarity value that is not zero. Just like we expected in the first exercise.

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.

Answer:
This is code is in python3 and tested on windows10. The sample data is reduced because of huge run time and pc hanging. Echo_assignment7_2.py

```
import random
import os
import sys
import re
import numpy as np
import matplotlib.pyplot as plt
import operator
from collections import Counter


def wordF(fileName):
    file=open(fileName,"r+", encoding="utf8")
    wordcount={}
    for word in file.read().split():
        word = word.lower()
        if word not in wordcount:
            wordcount[word] = 1
        else:
            wordcount[word] += 1
    return wordcount

def countFromFile(fileName):
    numberOfWords=0
    numberOfChar=0
    stat={}
    with open(fileName, 'r', encoding='utf8') as f:
        for contents in f:
            wordsList = contents.split()
            numberOfWords +=len(wordsList)
            for items in wordsList:
                numberOfChar+=len(items)
    stat[0] = numberOfChar
    stat[1] = numberOfWords
    return stat

def generateText(tp1,cdfdata,n,newFileName):
    lenOfDict = len(cdfdata)
    count = 0
    fullString = ""
    while (count<=n):
        r = random.random()
        for i in range(0,lenOfDict):
            if(r<=cdfdata[i]):
                fullString += str(tp1[i][0])
```

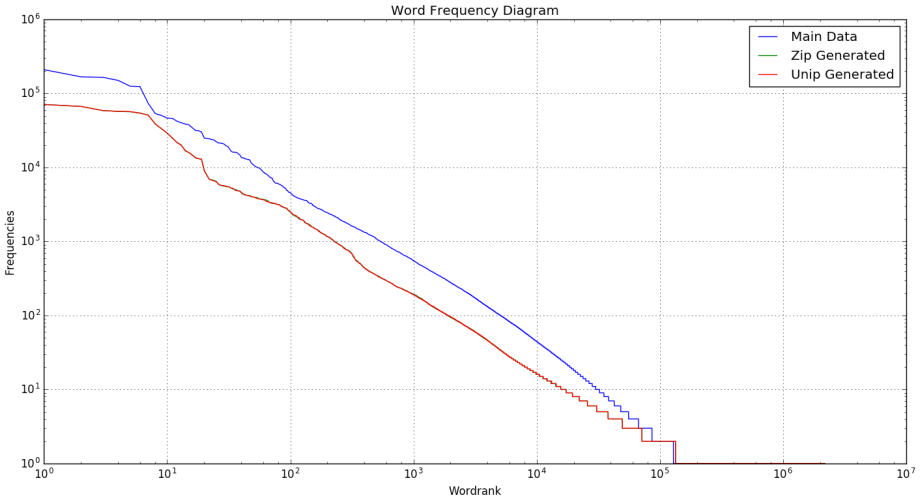
```
45:                 break
46:         count = count + 1
47:     with open(newFileName, 'a') as out:
48:         out.write(fullString)
49:
50: def drawHistogramMulti(sortedMainDataFreq, sortedZipDataFreq, sortedUnipDataFreq)
51:     plt.gca().set_color_cycle(['blue', 'green', 'red'])
52:     plt.plot(sortedMainDataFreq)
53:     plt.plot(sortedZipDataFreq)
54:     plt.plot(sortedUnipDataFreq)
55:     plt.legend(['Main Data', 'Zip Generated', 'Unip Generated'], loc='upper right
56:     plt.title('Word Frequency Diagram')
57:     plt.ylabel('Frequencies')
58:     plt.xlabel('Wordrank')
59:     plt.grid('on')
60:     plt.yscale('log')
61:     plt.xscale('log')
62:     mng = plt.get_current_fig_manager()
63:     mng.window.state('zoomed')
64:     plt.show()
65:
66: def drawCDF(sortedMainDataFreq, sortedZipDataFreq, sortedUnipDataFreq):
67:     cumsumMain=np.cumsum(sortedMainDataFreq)
68:     normedcumsumMain=[x/float(cumsumMain[-1]) for x in cumsumMain]
69:     cumsumZip=np.cumsum(sortedZipDataFreq)
70:     normedcumsumZip=[x/float(cumsumZip[-1]) for x in cumsumZip]
71:     cumsumUnip=np.cumsum(sortedUnipDataFreq)
72:     normedcumsumUnip=[x/float(cumsumUnip[-1]) for x in cumsumUnip]
73:
74:     C = [a - b for a, b in zip(normedcumsumMain, normedcumsumZip)]
75:     print('Maximum pointwise distance for zipf_probabilities: ',max(C))
76:     D = [a - b for a, b in zip(normedcumsumMain, normedcumsumUnip)]
77:     print('Maximum pointwise distance for uniform_probabilities : ',max(D))
78:
79:     plt.plot(normedcumsumMain)
80:     plt.plot(normedcumsumZip)
81:     plt.plot(normedcumsumUnip)
82:     plt.gca().set_color_cycle(['blue', 'green', 'red'])
83:     plt.legend(['Main Data', 'Zipf Generated', 'Uniform Generated'], loc='upper le
84:     plt.ylabel('CDF')
85:     plt.xlabel('Wordrank')
86:     plt.xscale('log')
87:     plt.title('CDF diagram for wordrank')
88:     mng = plt.get_current_fig_manager()
89:     mng.window.state('zoomed')
90:     plt.show()
91:
92: def main():
93:     #stat = countFromFile("asam.txt")
```

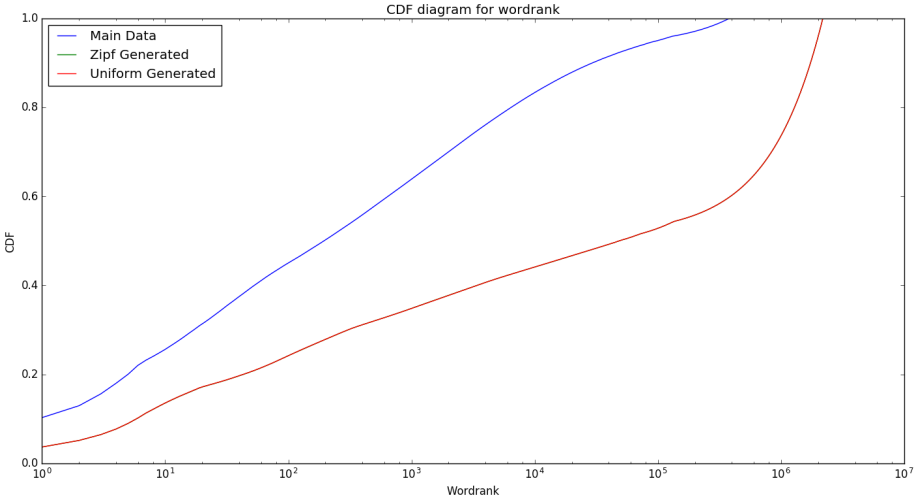
```
94:     #simplewiki.txt
95:     stat = countFromFile("simplewiki.txt")
96:     numberOfChar = stat[0]
97:     numberOfWords = stat[1]
98:
99:     zifp = {' ': 0.17840450037213465, '1': 0.004478392057619917, '0': 0.003671824
100:            '_': 2.3580656240324293e-05, 'a': 0.07264712490903191, 'c': 0.0256376
101:            'm': 0.022165129421030945, 'l': 0.03389465109649857, 'o': 0.057928476
102:            'y': 0.013084644140464391, 'x': 0.0014600810295164054, 'z': 0.0010488
103:    #for another probability set, let it be x1
104:    unip = {' ': 0.1875, 'a': 0.03125, 'c': 0.03125, 'b': 0.03125, 'e': 0.03125,
105:            'j': 0.03125, 'm': 0.03125, 'l': 0.03125, 'o': 0.03125, 'n': 0.03125,
106:            'v': 0.03125, 'y': 0.03125, 'x': 0.03125, 'z': 0.03125}
107:
108:    sorted_zifp = sorted(zifp.items(), key=operator.itemgetter(0)) #Make tuples
109:    sorted_unip = sorted(zifp.items(), key=operator.itemgetter(0)) #Make tuples
110:
111:    lenOfDict = len(sorted_zifp)
112:    lenOfDict_unip = len(sorted_unip)
113:
114:    zifpCDFList = []
115:    unipCDFList = []
116:
117:    for i in range(0,lenOfDict):
118:        if(i == 0):
119:            zifpCDFList.append(sorted_zifp[i][1])
120:        else:
121:            zipCDFap = zifpCDFList[i-1] + sorted_zifp[i][1]
122:            zifpCDFList.append(zipCDFap)
123:
124:    for i in range(0,lenOfDict_unip):
125:        if(i == 0):
126:            unipCDFList.append(sorted_unip[i][1])
127:        else:
128:            unipCDFap = unipCDFList[i-1] + sorted_unip[i][1]
129:            unipCDFList.append(unipCDFap)
130:
131:    generateText(sorted_zifp,zifpCDFList,numberOfChar,"ziptext.txt")
132:    generateText(sorted_unip,unipCDFList,numberOfChar,"uniptext.txt")
133:
134:    #mainDataFreq = wordF('asam.txt')
135:    mainDataFreq = wordF('simplewiki.txt')
136:    zipDataFreq = wordF('ziptext.txt')
137:    unipDataFreq = wordF('uniptext.txt')
138:
139:    list(mainDataFreq.values())
140:
141:    sortedMainDataFreq = list(mainDataFreq.values())
142:    sortedMainDataFreq = sorted(sortedMainDataFreq, reverse=True)
```

```
143:
144:     sortedZipDataFreq = list(zipDataFreq.values())
145:     sortedZipDataFreq = sorted(sortedZipDataFreq, reverse=True)
146:
147:     sortedUnipDataFreq = list(unipDataFreq.values())
148:     sortedUnipDataFreq = sorted(sortedUnipDataFreq, reverse=True)
149:
150:     drawHistogramMulti(sortedMainDataFreq, sortedZipDataFreq, sortedUnipDataFreq)
151:
152:     drawCDF(sortedMainDataFreq, sortedZipDataFreq, sortedUnipDataFreq)
153:
154: if __name__ == '__main__':
155:     main()
```

For 2.1 Generator is built in the code.

For 2.2 The diagram is shown in the code. Also given bellow. The differences of generated data is in micro level.So it seems like a single line.





For 2.3, the result for k-s test is given bellow for first time and second time.

Firt Time

```
Maximum pointwise distance for zipf_probabilities:  0.424275719269
Maximum pointwise distance for uniform_probabilities :  0.424177280714
```

Second Time

```
Maximum pointwise distance for zipf_probabilities:  0.424351348201
Maximum pointwise distance for uniform_probabilities :  0.424259275966
```

I would use uniform probabilities. Though both of the results are almost same but for uniform probabilities(.4241). it is better than zipf probabilities(.4242).
The result changes a little for second time running. The test result is given bellow. But it remains same in the case that Uniform probabilities is little better than Zipf probabilities for this result of this data set. Zipf:0.424351348201, Uniform:0.424259275966

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:
The whole code is given bellow. It is written in python3 and tested on windows10. (Echo_assignment7_3.py)

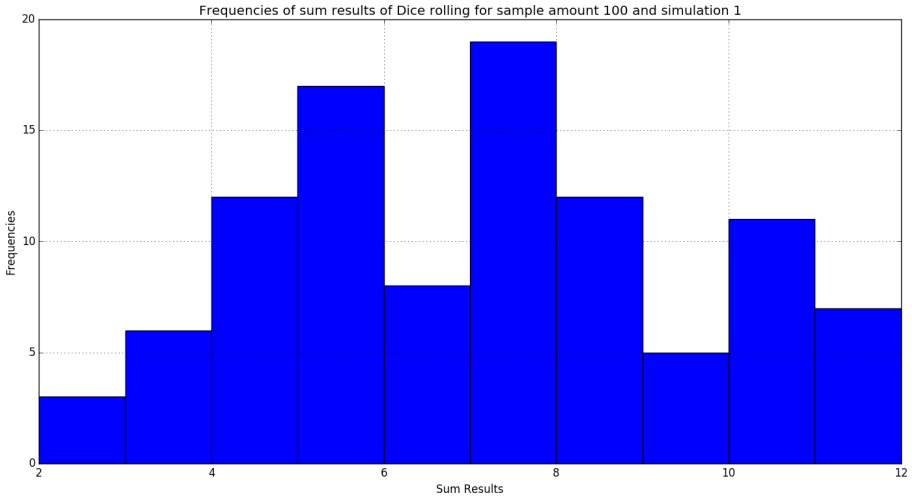
```
 1: import random
 2: import numpy as np
 3: import matplotlib.pyplot as plt
 4:
 5: def dice_roll():
 6:     global dice1Result
 7:     dice1Result.append(random.randint(1,6))
 8:     global dice2Result
 9:     dice2Result.append(random.randint(1,6))
10:
```

```
11: def drawHistogram(sumDiceResult,n,rn):
12:     plt.hist(sumDiceResult)
13:     plt.title('Frequencies of sum results of Dice rolling')
14:     plt.title('Frequencies of sum results of Dice rolling for sample amount %d and
15:     plt.ylabel('Frequencies')
16:     plt.xlabel('Sum Results')
17:     plt.grid('on')
18:     #plt.switch_backend('TkAgg')
19:     mng = plt.get_current_fig_manager()
20:     mng.window.state('zoomed')
21:     plt.show()
22:
23: #http://stackoverflow.com/questions/3209362/how-to-plot-empirical-cdf-in-matplotli
24: def drawCDF(sumDiceResult,n,rn):
25:     sorted1 = np.sort(sumDiceResult)
26:     theMedian = np.median(sorted1)
27:
28:     print("Median: ", theMedian)
29:     totalNum9 = sum(i <= 9 for i in sorted1)
30:     totalNum9per = ((totalNum9*100)/n)/100
31:     print("Probability of dice sum to be equal or less than 9: ",totalNum9per)
32:     yvals = np.arange(len(sorted1))/float(len(sorted1))
33:     plt.plot(sorted1, yvals)
34:     plt.plot((0, 12), (.5, .5))
35:     plt.plot((9, 9), (0, 1))
36:     plt.plot((theMedian,theMedian), (0, .5))
37:     plt.plot((0,12), (totalNum9per, totalNum9per))
38:     plt.legend(['CDF', '50%', '<=9','Median','Percentage of <=9',], loc='lower rig
39:     plt.gca().set_color_cycle(['blue', 'green', 'red', 'orange','yellow'])
40:     plt.annotate('Median: %d'%theMedian, (theMedian,.5),xytext=(0.7, 0.7),arrowpro
41:     plt.annotate('Probability (<=9): %s'%totalNum9per, (9,totalNum9per),xytext=(1
42:     plt.title('CDF of dice sum frequency for sample amount %d and simulation %d'%
43:     plt.ylabel('Probability')
44:     plt.xlabel('Sum Results (x)')
45:     plt.grid('on')
46:
47:     #plt.switch_backend('TkAgg')
48:     mng = plt.get_current_fig_manager()
49:     mng.window.state('zoomed')
50:     plt.show()
51:
52: def pointDistance(listSumOf1, listSumOf2, n):
53:     cumsumS1=np.cumsum(listSumOf1)
54:     normedcumsumS1=[x/float(cumsumS1[-1]) for x in cumsumS1]
55:     cumsumS2=np.cumsum(listSumOf2)
56:     normedcumsumS2=[x/float(cumsumS2[-1]) for x in cumsumS2]
57:     D = [a - b for a, b in zip(normedcumsumS1, normedcumsumS2)]
58:     print('Maximum point distance of CDFs generated from simulation 1 and 2 for '
59:     print('\n')
```

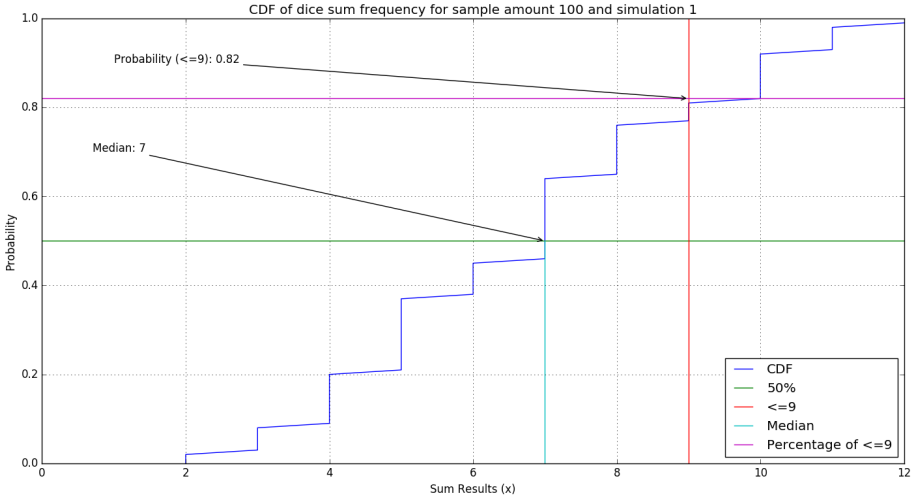
```
60: def main():
61:     global dice1Result
62:     dice1Result = []
63:     global dice2Result
64:     dice2Result = []
65:     global sumDiceResult
66:     sumDiceResult = []
67:     global sumDiceResult2
68:     sumDiceResult2 = []
69:
70:     #Simulation for 100 samples ###################################################
71:
72:     #Simulation 1 for 100 ###################
73:     for x in range(0, 100):
74:         dice_roll()
75:     sumDiceResult = [x + y for x, y in zip(dice1Result, dice2Result)]
76:     print('\nResults for simulation 1 for 100 samples')
77:     #Draw histogram
78:     drawHistogram(sumDiceResult,100,1)
79:     #Draw CDF
80:     drawCDF(sumDiceResult,100,1)
81:
82:     #Simulation 2 for 100 ###################
83:     dice1Result = []
84:     dice2Result = []
85:     for x in range(0, 100):
86:         dice_roll()
87:     sumDiceResult2 = [x + y for x, y in zip(dice1Result, dice2Result)]
88:     print('\nResults for simulation 2 for 100 samples')
89:     #Draw histogram
90:     drawHistogram(sumDiceResult2,100,2)
91:     #Draw CDF
92:     drawCDF(sumDiceResult2,100,2)
93:     pointDistance(sumDiceResult, sumDiceResult2,100)
94:
95:     #Simulation for 1000 samples ##################################################
96:     sumDiceResult = []
97:     sumDiceResult2 = []
98:     dice1Result = []
99:     dice2Result = []
100:
101:    #Simulation 1 for 1000 ###################
102:    for x in range(0, 1000):
103:        dice_roll()
104:    sumDiceResult = [x + y for x, y in zip(dice1Result, dice2Result)]
105:    print('\nResults for simulation 1 for 1000 samples')
106:    #Draw histogram
107:    drawHistogram(sumDiceResult,1000,1)
108:    #Draw CDF
```

```
109:     drawCDF(sumDiceResult,1000,1)
110:     #Simulation 2 for 1000 ###################
111:     dice1Result = []
112:     dice2Result = []
113:     for x in range(0, 1000):
114:         dice_roll()
115:     sumDiceResult2 = [x + y for x, y in zip(dice1Result, dice2Result)]
116:     print('\nResults for simulation 2 for 1000 samples')
117:     #Draw histogram
118:     drawHistogram(sumDiceResult2,1000,2)
119:     #Draw CDF
120:     drawCDF(sumDiceResult2,1000,2)
121:     pointDistance(sumDiceResult, sumDiceResult2,1000)
122:
123: if __name__ == '__main__':
124:     main()
```

1. Picture is given bellow from first simulation (n=100).



2. CDF - from first simulation(n=100).



3. The plot is marked in avobe figures. The results are:
Median of dice sum is 7
The probabilities for dice sum to be less than or equal to 9 is: 0.82
Results for 3, 4 and 5 is shown in the figure of following question num 4.
4 and 5.
After running 2 times for 100 data the maximum point wise distance for both CDFs is: 0.0139760252813
After running 2 times for 1000 data the maximum point wise distance for both CDFs is: 0.00438882204706

```
D:\MScWebScience\IntroToWS\assignments\Echo\Echo\assignment7_WorkingFolder\task3>python3 Echo_assignment7_3.py

Results for simulation 1 for 100 samples
Median:  7.0
Probability of dice sum to be equal or less than 9:  0.82
C:\Users\nawaz\AppData\Local\Programs\Python\Python35-32\lib\site-packages\matplotlib\cbook.py:137: MatplotlibDeprecationWarning: The set_color_cycle attribute was depr
ecated in version 1.5. Use set_prop_cycle instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)

Results for simulation 2 for 100 samples
Median:  7.0
Probability of dice sum to be equal or less than 9:  0.85
Maximum point distance of CDFs generated from simulation 1 and 2 for  100  data:  0.0139760252813



Results for simulation 1 for 1000 samples
Median:  7.0
Probability of dice sum to be equal or less than 9:  0.848

Results for simulation 2 for 1000 samples
Median:  7.0
Probability of dice sum to be equal or less than 9:  0.842
Maximum point distance of CDFs generated from simulation 1 and 2 for  1000  data:  0.00438882204706
```

6. For 100 sample the distance is 0.013 and for 1000 sample the distance is 0.004. The conclusion is, if we take more sample the similarity increases. The cdf becomes closer. Better resulation figures can be found here: In this link

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`.