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: Golf

Members: Atique Baig, Mtarji Adam, Deepak Garg

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:

In our opinion, D2 and D3 are the most similar because from a semantic perspective, they have the closest meaning. But from a word by word perspective, they have no words in common, only D1 and D2 have common words but have different meanings.

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?

   Answer:

   In order to model these documents, we need a language ”space” from which we can constitute these documents again. the number of base vectors is the minimum number of elements that can generate these 3 documents, considering a word as an element, we will need 19 base vector for our model.

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

   Answer:

   Each dimension of our vector space is a word base vector.

3. How many dimensions does the vector space have?

   Answer:

   The vector space has as many dimensions as base vectors, which is 19.

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

   Answer:

Word	vector
$\overrightarrow{this}$	$(1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$
$\vec{is}$	$(0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$
$\vec{a}$	$(0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$
$\overrightarrow{text}$	$(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$
$\overrightarrow{about}$	$(0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$
$\overrightarrow{web}$	$(0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0)$
$\overrightarrow{science}$	$(0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0)$
$\overrightarrow{covering}$	$(0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0)$
$\overrightarrow{the}$	$(0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0)$
$\overrightarrow{analysis}$	$(0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0)$
$\vec{of}$	$(0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0)$
$\overrightarrow{corpora}$	$(0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0)$
$\overrightarrow{scientific}$	$(0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0)$
$\overrightarrow{methods}$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0)$
$\overrightarrow{are}$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0)$
$\overrightarrow{used}$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0)$
$\vec{to}$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0)$
$\overrightarrow{analyze}$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0)$
$\overrightarrow{webpages}$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)$

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.

   Answer:

   Having a vector space
   $V = \{\overrightarrow{this}, \vec{is}, \vec{a}, \overrightarrow{text}, \overrightarrow{about}, \overrightarrow{web}, \overrightarrow{science}, \overrightarrow{covering}, \overrightarrow{the}, \overrightarrow{analysis}, \vec{of}, \overrightarrow{corpora}, \overrightarrow{scientific}, \overrightarrow{methods}, \overrightarrow{are}, \overrightarrow{used}, \vec{to}, \overrightarrow{analyze}, \overrightarrow{webpages}\}$

   Our document vector would be the following:

$$\begin{aligned}\overrightarrow{D_1} &= \sum_{1}^{19} tf(\vec{w}_i, D_1)\,\vec{w}_i \\ &= \overrightarrow{this} + \overrightarrow{is} + \vec{a} + \overrightarrow{text} + \overrightarrow{about} + \overrightarrow{web} + \overrightarrow{science} \\ &= (1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0)\end{aligned} \tag{1}$$

$$\begin{aligned}\overrightarrow{D_2} &= \sum_{1}^{19} tf(\vec{w}_i, D_2)\,\vec{w}_i \\ &= \overrightarrow{web} + \overrightarrow{science} + \overrightarrow{is} + \overrightarrow{covering} + \overrightarrow{the} + \overrightarrow{analysis} + \overrightarrow{of} \\ &+ \overrightarrow{text} + \overrightarrow{corpora} \\ &= (0,0,1,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0)\end{aligned} \tag{2}$$

$$\begin{aligned}\overrightarrow{D_3} &= \sum_{1}^{19} tf(\vec{w}_i, D_3)\,\vec{w}_i \\ &= \overrightarrow{scientific} + \overrightarrow{methods} + \overrightarrow{are} + \overrightarrow{used} + \overrightarrow{to} + \overrightarrow{analyze} \\ &+ \overrightarrow{webpages} \\ &= (0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1)\end{aligned} \tag{3}$$

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

   Answer:

   First, lets calculate the distance (in this case length of document vectors).

$$\begin{aligned}||\overrightarrow{D_1}|| &= \sqrt{1+1+1+1+1+1+1} \\ &= \sqrt{7}\end{aligned} \tag{4}$$

$$\begin{aligned}||\overrightarrow{D_2}|| &= \sqrt{1+1+1+1+1+1+1+1+1} \\ &= \sqrt{9}\end{aligned} \tag{5}$$

$$\begin{aligned}||\overrightarrow{D_3}|| &= \sqrt{1+1+1+1+1+1+1} \\ &= \sqrt{7}\end{aligned} \tag{6}$$

Lets calculate the cosine similarity of each pair:

$$\begin{aligned}\theta_{12} &= Cos^{-1}\left(\frac{\langle\overrightarrow{D_1},\overrightarrow{D_2}\rangle}{||\overrightarrow{D_1}|| \times ||\overrightarrow{D_2}||}\right)\\ &= Cos^{-1}\left(\frac{tf(text)+tf(is)+tf(web)+tf(science)}{||\overrightarrow{D_1}|| \times ||\overrightarrow{D_2}||}\right)\\ &= \frac{4}{\sqrt{7}\times\sqrt{9}}\\ &= 1.04\end{aligned} \tag{7}$$

$$\begin{aligned}\theta_{13} &= Cos^{-1}\left(\frac{\langle\overrightarrow{D_1},\overrightarrow{D_3}\rangle}{||\overrightarrow{D_1}|| \times ||\overrightarrow{D_3}||}\right)\\ &= Cos^{-1}\left(\frac{0}{||\overrightarrow{D_1}|| \times ||\overrightarrow{D_3}||}\right)\\ &= 1.57\end{aligned} \tag{8}$$

$$\begin{aligned}\theta_{23} &= Cos^{-1}\left(\frac{\langle\overrightarrow{D_2},\overrightarrow{D_3}\rangle}{||\overrightarrow{D_2}|| \times ||\overrightarrow{D_3}||}\right)\\ &= Cos^{-1}\left(\frac{0}{||\overrightarrow{D_2}|| \times ||\overrightarrow{D_3}||}\right)\\ &= 1.57\end{aligned} \tag{9}$$

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

   Answer:

   Since the cosine similarity between D_1 and D_2 gave us the lowest angle of 1.04, it is by definition, these 2 are the 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.

Answer:

The results of the model does not match our expectation, this is mainly due to this model using keywords to find similarity but not taking into consideration the semantic. If for example, we can use a model that goes further and checks for every word vector, other close word vectors in the same space, the model might be able to recognize for example that science and scientific are close (maybe using a predefined database of such information).

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.

Figure 1: Example of a generate text using the zipf distribution

```
ythomnhge s mee tinbofe ceuf de  sg hsdd  obdepeie r  se shuleh ssoa hgse
esno  eeesewotefea s 5easnesaetocb  cs  trp  nou  eecoaastoaijtn hdnodo
aste v ccngmfmcvlae p peituhtstshien egho3 t loae u uoaoltvmlbpd ee 2  9
tlnystut  erraaie 5hne t t ngt wltrtlllt ls gry t e hi aw19c tiesnnrathhc
thdeutptiwetlieou r osta iddacbewho  as mlcebpimoditpooina o0uonianu srot
erniiee dn heli  h9l ouhi g lfatholpse oiewh eeih rssmb  n nipjsatt ovnlr
en a obaoahaw  dceeleoc oidafntncohlnrreischea iasia snhtoddeeiaamnet  ah
upt  7luet imaihgsb es mlfalxtobtrgtyaaeusylueefr ec b  mlese o2e oldno a
dbucrtbef  bfnecns 2 li1ru ewr o  np s uma 1nxnctdde dled1oeeesskindtneua
siipoeeeg etret ghchina  splraat nsuomi tieu  cpawr  nis cccl 9nleanchoti
5cceorubioamoosacari nili n    li gaoaa lh ib3yi c3li aosctdsmcedltssgrse
iupho thfpndee ds amadgle rhy mbal ebrocsetuuecb  efvehuesw1h lninaee a r
snr nnsaunn l  ol eirs ev eim o   gn tdrr  t ecldkp e hi scsd tneawien n
hiesaammvyuuihlteaioofullceiooefcoopstrnf1sxrytnc tiss wlylaia nl rdi8l
ohuioonhi yanaecrsnlh ritiasdea rld tcysc e a mw   rsek ciiewv habwjr eic
e5nod lfssoseicn  ysa cayansd cs i iaaha c rr emsnkspfnemtreave3eedycdn g
ike mmus aeasar epa ly atcth ameppd o iur sllt e    die tzi kaerr idiorlv
nn cnnnttba aiuaus neomaehfctcsar mandani  sdssrloe  rhaso  ctssiadsttsac
```

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.

Figure 2: frequency by Word ranking using loglog plot

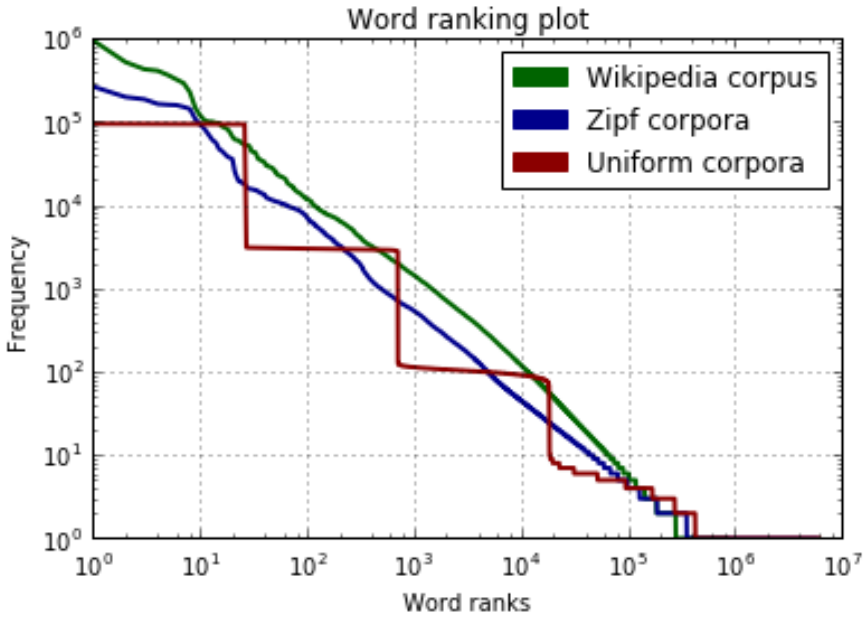
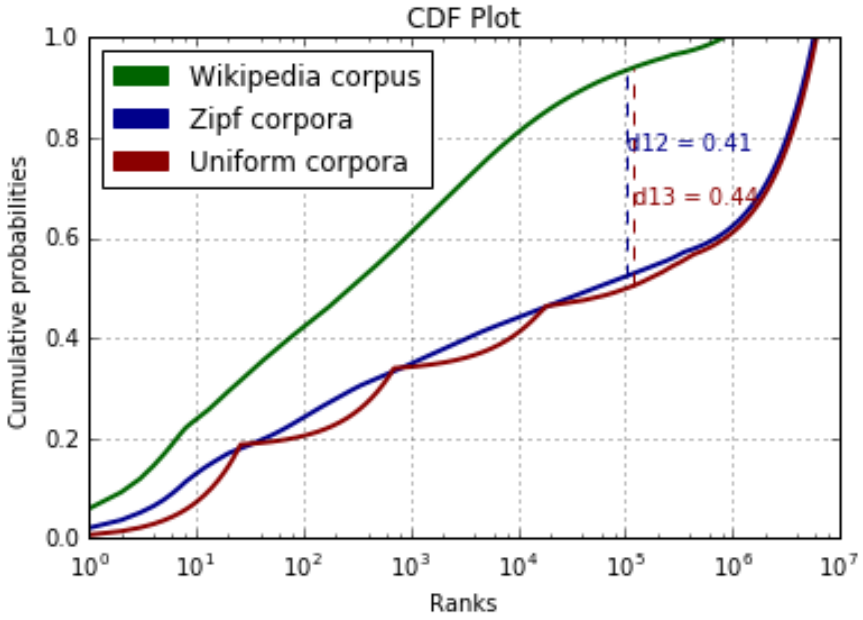


Figure 3: CDF plot and maximum distance



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.

Answer:

Judging by the maximum point wise distance between the text corpus and the generated corpora, we notice that the zipf distribution is better fitted to our model (lowest max distance at 0.41 for 0.44) and thus, would be a better choice the the uniform, which already shows big distances on multiple ranks on the ranking log log plot.

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:

A second simulation gives us similar results with zipf as the best fit (lowest max distance of 0.4 for 0.45).

generator.py:

```
"""
Introduction to Web Science
Assignment 7
Question 2
Team : golf

"""
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import numpy as np
import probabilities as dist
import string
import operator
import random
from collections import Counter

#9 function to calculate the cumulative probabilities of a distribution
def generate_cumulative_probs(distribution):
    sorted_keys = sorted(distribution,key=distribution.get)
```

```
    cumul_values = np.cumsum(sorted(distribution.values()))
    cumulative_probabilities = dict(zip(sorted_keys, cumul_values))
    return cumulative_probabilities

# sample a random character fom a cumulative distribution
def sampleCharacter(cumul):
    r= random.random()
    x_condidates={}#we store all values > then the random number
    for key, value in cumul.items():
        if value > r:
            x_condidates[key]=value
    # we pick the key of the lowest value
    character = min(x_condidates, key=x_condidates.get)
    return character

def fetchAllWords(filename):
    f = open(filename,'r',encoding='utf8')
    allWords=[]
    for line in f:
        line = line[:-1]
        words= line.split()
        allWords.extend(words)
    return allWords

# returns the ranks, frequencies and normelized cumusum
def getWordStats(filename):
    words,frequencies=zip(*Counter(fetchAllWords(filename)).most_common())
    cumsum=np.cumsum(frequencies)
    normedcumsum=[x/float(cumsum[-1]) for x in cumsum]
    wrank = {words[i]:i+1 for i in range(0,len(words))}
    return words,wrank,frequencies,normedcumsum

"""
Starting the script
"""
# generating cumulative probabilities
uniform_set=generate_cumulative_probs(dist.uniform_probabilities)
zipf_set=generate_cumulative_probs(dist.zipf_probabilities)

# we are only looking for alphabets and spaces
wanted_chars=list(string.ascii_lowercase)
wanted_chars.append(' ')
# we create a dictionary to store our counts
char_count = dict((x,0) for x in wanted_chars)
```

```
# using the artivle per line dump of simple wikipedia
with open('article-per-line.txt','r',encoding="utf8") as f:
    for c in list(f.read()):
        if c in char_count:
            char_count[c] += 1

# sum of all counts
n=sum(char_count.values())

print('Total characters and spaces found = ',n)

# global variable to store our generated uniform dataset
gen_uniform_dataset=''

for i in range(n):
    # we sample n times
    gen_uniform_dataset+=sampleCharacter(uniform_set)

with open('uniform_dataset.txt','w') as uf:
    uf.write(gen_uniform_dataset)

gen_zipf_dataset=''

for i in range(n):
    gen_zipf_dataset+=sampleCharacter(zipf_set)

with open('zipf_dataset.txt','w') as uf:
    uf.write(gen_zipf_dataset)

# getting the ranks, frequencies and normilzed cumsums for ploting
words_1,wrank_1,frequencies_1,normedcumsum_1=getWordStats('article-per-line.txt')
words_2,wrank_2,frequencies_2,normedcumsum_2=getWordStats('zipf_dataset.txt')
words_3,wrank_3,frequencies_3,normedcumsum_3=getWordStats('uniform_dataset.txt')

"""
ploting word rankings
"""
plt.figure()
r1 = np.arange(1, len(words_1)+1)
f1=np.array([float(i) for i in frequencies_1])

r2 = np.arange(1, len(words_2)+1)
f2=np.array([float(i) for i in frequencies_2])
```

```
r3 = np.arange(1, len(words_3)+1)
f3=np.array([float(i) for i in frequencies_3])

# using a loglog plot for readability
plt.loglog(r1, f1,
           color='darkgreen',
           linewidth = 2)
plt.loglog(r2, f2,
           color='darkblue',
           linewidth = 2)
plt.loglog(r3, f3,
           color='darkred',
           linewidth = 2)

text_corupus_patch = mpatches.Patch(color='darkgreen',
                                    label='Wikipedia corpus')
zipf_corupus_patch = mpatches.Patch(color='darkblue',
                                    label='Zipf corpora')
uniform_corupus_patch = mpatches.Patch(color='darkred',
                                       label='Uniform corpora')
plt.legend(handles=[text_corupus_patch,zipf_corupus_patch,
                    uniform_corupus_patch])

plt.grid(True)
plt.xlabel("Word ranks")
plt.ylabel("Frequency")
plt.title('Word ranking plot')

plt.show()

"""
ploting CDF of word ranks
"""

n1=np.array([float(i) for i in normedcumsum_1])

n2=np.array([float(i) for i in normedcumsum_2])

n3=np.array([float(i) for i in normedcumsum_3])

plt.plot(r1, n1,
           color='darkgreen',
           linewidth = 2)
```

```
plt.plot(r2, n2,
          color='darkblue',
          linewidth = 2)
plt.plot(r3, n3,
          color='darkred',
          linewidth = 2)

plt.semilogx()#scale only x to log

# we are looking to draw distance lines on plots from each generated model
# to the corpus model
# [NEED_OPTIMIZATION] we calculate the max of all distances
max_distance_1_2=max([abs(x) for x in list(map(operator.sub,
                      normedcumsum_1, normedcumsum_2))])
#used to store y coordinates
y12s=[]
# rank used
x12=0
# we iterate over each rank, once we find the one giving the maximum distance
# we save it in x12
# NOTE: this part need a more pythonic and optimized solution
# we check using the intersection of ranks between the sets
for r in [val for val in r1 if val in r2]:
    y12=normedcumsum_1[r]
    y21=normedcumsum_2[r]
    if max_distance_1_2==abs(y12-y21):
        x12=r
        y12s.append(y12)
        y12s.append(y21)
        break
plt.axvline(x=x12, ymin=min(y12s), ymax=max(y12s),
            linestyle ='dashed',color='darkblue')
plt.text(x12+2, sum(y12s)/2+0.05,
         'd12 = %.2f'%max_distance_1_2,color='darkblue')

# we do the same for dataset 1 and 3
max_distance_1_3=max([abs(x) for x in list(map(operator.sub,
                      normedcumsum_1, normedcumsum_3))])

y13s=[]
x13=0
for r in [val for val in r1 if val in r3]:
    y13=normedcumsum_1[r]
    y31=normedcumsum_3[r]
```

```
    if max_distance_1_3==abs(y13-y31):
        x13=r
        y13s.append(y13)
        y13s.append(y31)
        break
plt.axvline(x=x13, ymin=min(y13s), ymax=max(y13s),
            linestyle ='dashed',color='darkred')
plt.text(x13+2, sum(y13s)/2-0.05,
'd13 = %.2f'%max_distance_1_3,color='darkred')


text_corupus_patch = mpatches.Patch(color='darkgreen',
                                    label='Wikipedia corpus')
zipf_corupus_patch = mpatches.Patch(color='darkblue',
                                    label='Zipf corpora')
uniform_corupus_patch = mpatches.Patch(color='darkred',
                                       label='Uniform corpora')
plt.legend(handles=[text_corupus_patch,
                    zipf_corupus_patch,uniform_corupus_patch],
                    bbox_to_anchor=(0, 1), loc='upper left', ncol=1)

plt.grid(True)
plt.xlabel("Ranks")
plt.ylabel("Cumulative probabilities")
plt.title('CDF Plot')



plt.show()
```

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.

   Answer:

Figure 4: Histogram using n=100

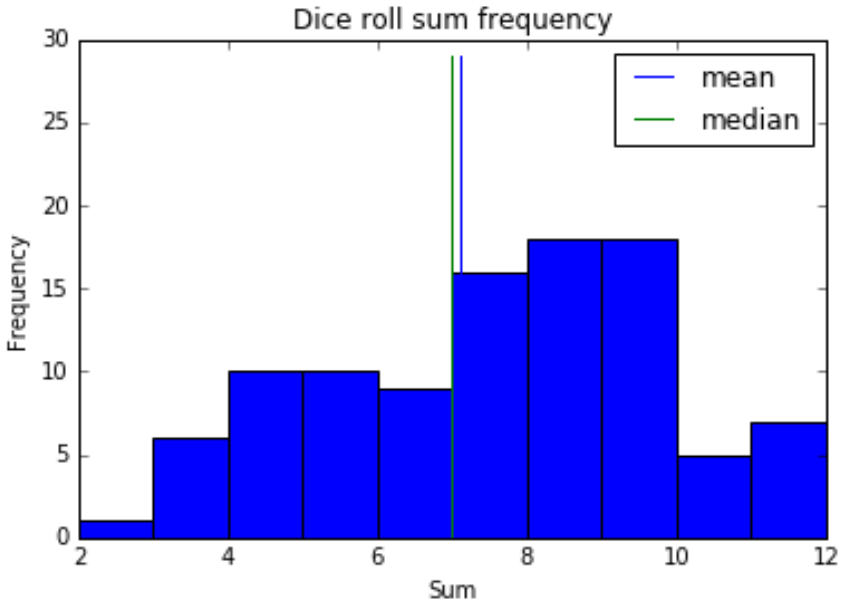
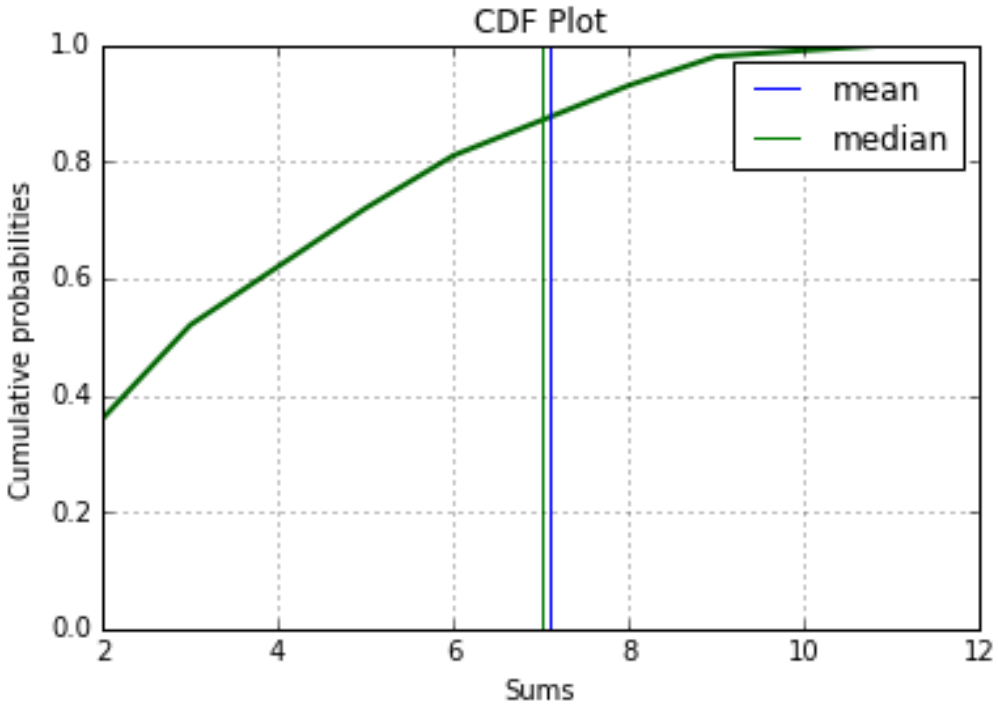


Figure 5: CDF using n=100

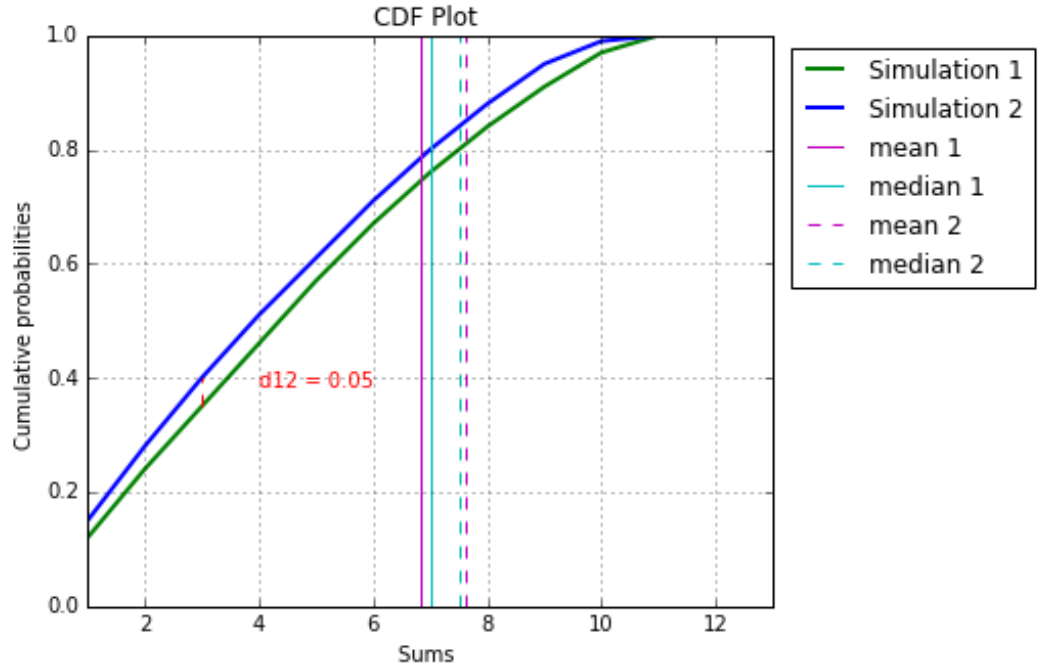



```
Median: 7.0
Mean: 7.1
```

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?

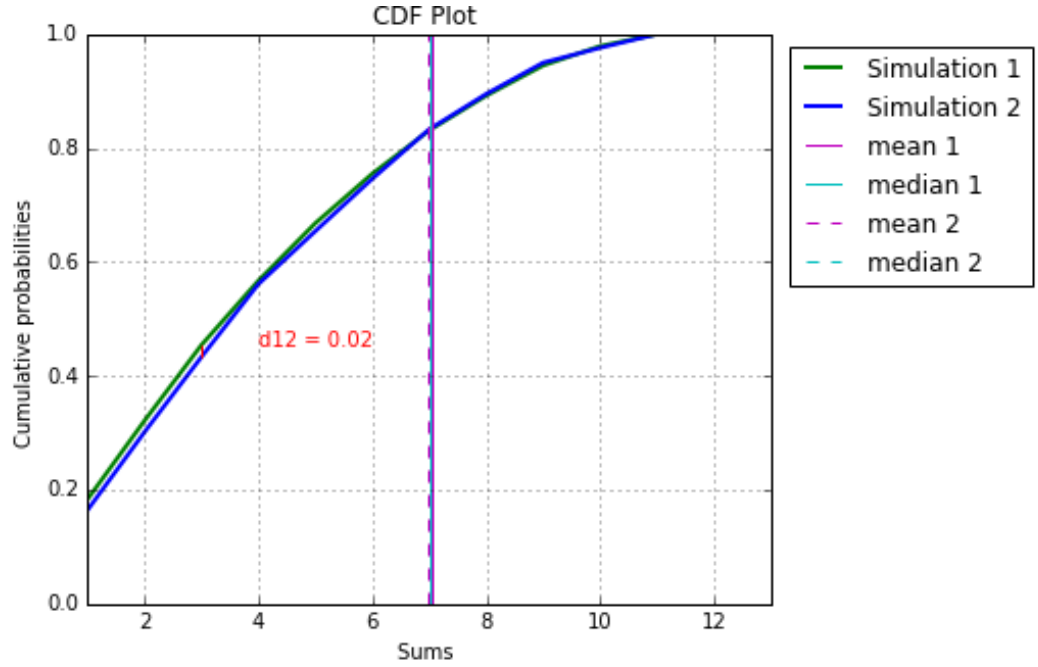
Answer:

Figure 6: CDF using n=100 and 2 simulations




```
Median 1st simulation: 7.0
Mean 1st simulation: 6.82
Median 2nd simulation: 7.5
Mean 2nd simulation: 7.61
Maximum distance is : 0.05
```

Figure 7: CDF using n=1000 and 2 simulations




```
Median 1st simulation: 7.0
Mean 1st simulation: 7.047
Median 2nd simulation: 7.0
Mean 2nd simulation: 6.969
Maximum distance is : 0.021
```

After doing 2 simulation using n=100 and 2 simulation using n=1000, we can observe that high number of rolls (n) gives a better and consistent accuracy for our model, with lower max point wise distance and closer medians and means.

Scripts:

dice.py:

```
"""
Introduction to Web Science
Assignment 7
Question 3
Team : golf

"""
import matplotlib.pyplot as plt
import numpy as np
import operator
import random
from collections import Counter

# function to roll a fair sided dice
def rollDice():
    # equall probabilty to get a number between 1 and 6 (7 excluded)
    outcome=random.randrange(1,7)
    print(outcome)
    return outcome

# get the CDF from an array
def getCDF(sums):
    dsums,frequencies=zip(*Counter(sums).most_common())
    cumsum=np.cumsum(frequencies)
    normedcumsum=[x/float(cumsum[-1]) for x in cumsum]
    dsrank = {dsums[i]:i+1 for i in range(0,len(dsums))}
    return dsums,dsrank,frequencies,normedcumsum

# simulate the model taking n as a paramter
def simulation(n):
    sums=[]
    for i in range(n):
        d1 = rollDice()
        d2 = rollDice()
        dsum = d1+d2
```

```
        sums.append(dsum)
    return sums

n=1000
sums1=simulation(n)
sums2=simulation(n)

median1 = np.median(sums1)
median2 = np.median(sums2)

mean1 = np.mean(sums1)
mean2 = np.mean(sums2)

"""
Ploting histogram of the first simulation (n=100 or n=1000)
"""
plt.hist(sums1,bins=range(0, 13, 1))
plt.plot([mean1]*len(range(0,160)),range(0,160),"b",label="mean")
plt.plot([median1]*len(range(0,160)),range(0,160),"g",label="median")
plt.ylabel("Frequency")
plt.xlabel("Sum")
plt.title("Dice roll sum frequency for 1st simulation")
plt.xlim(2,12)
plt.legend()
plt.show()

dsums1,dsrank1,frequencies1,normedcumsum1=getCDF(sums1)
dsums2,dsrank2,frequencies2,normedcumsum2=getCDF(sums2)

"""
ploting CDF for both simulations using n = 100 or 1000
"""
plt.figure()
fig = plt.gcf()
fig.set_size_inches(6,5)
fig.savefig('test2png.png', dpi=100)

r1 = np.arange(1, len(dsums1)+1)
s1 = np.array([float(i) for i in normedcumsum1])

plt.plot(r1, s1, color='green',label='Simulation 1' ,linewidth = 2)

r2 = np.arange(1, len(dsums2)+1)
s2 = np.array([float(i) for i in normedcumsum2])
```

```
plt.plot(r2, s2,
          color='blue',label='Simulation 2',
          linewidth = 2)

plt.grid(True)
plt.xlabel("Sums ")
plt.ylabel("Cumulative probabilities ")
plt.plot([mean1]*len(range(0,2)),range(0,2),"m",label="mean 1")
plt.plot([median1]*len(range(0,2)),range(0,2),"c",label="median 1")
plt.plot([mean2]*len(range(0,2)),range(0,2),"m",label="mean 2",
         linestyle ='dashed')
plt.plot([median2]*len(range(0,2)),range(0,2),"c",label="median 2",
         linestyle ='dashed')

max_distance_1_2=max([abs(x) for x in list(map(operator.sub,
                      normedcumsum1, normedcumsum2))])
#used to store y coordinates
y12s=[]
x12=0

comun_sums=[val for val in r1 if val in r2]

for r in comun_sums:
    y12=normedcumsum1[r]
    y21=normedcumsum2[r]
    if max_distance_1_2==abs(y12-y21):
        x12=r+1
        y12s.append(y12)
        y12s.append(y21)
        break

plt.axvline(x=x12, ymin=min(y12s), ymax=max(y12s),
            linestyle ='dashed',color='red')
plt.text(x12+1, sum(y12s)/2+0.01,
         'd12 = %.2f'%max_distance_1_2,color='red')

plt.title('CDF Plot ')
plt.xlim(1,13)
plt.legend(bbox_to_anchor=(0, 1 ,1.4, 0),  ncol=1)
plt.show()
print ("Median 1st simulation:",median1)
print ("Mean 1st simulation:",mean1)
print ("Median 2nd simulation:",median2)
```

```
print ("Mean 2nd simulation:",mean2)
print ("Maximum distance is :",max_distance_1_2)
```

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)?

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`.