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: Tango

1. Mariya Chkalova
   mchkalova@uni-koblenz.de
2. Arsenii Smyrnov
   smyrnov@uni-koblenz.de
3. Simon Schauß
   sschauss@uni-koblenz.de
4. Lukas Härtel
   lukashaertel@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 D_2 and D_3 are closest to each other, as they deal with the matter of web science, as opposed to the D_1, which is a plain self descrbing statement not mentioning any more information.

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.

Answers

1. These documents would require the base vectors v_{this}, v_{is}, v_a, v_{text}, v_{about}, v_{web}, $v_{science}$, $v_{covering}$, v_{the}, $v_{analysis}$, v_{of}, $v_{corpora}$, $v_{scientific}$, $v_{methods}$, v_{are}, v_{used}, v_{to}, $v_{analyze}$, $v_{webpages}$. This makes 19 base vectors.
2. A document is a linear combination of the base vectors, where the factor of a base vector stands for the number of word occurences in the document. Therefore, the dimension shows how often a word was used in a document.
3. The base vectors are linearly independent, therefore the dimension is the same as the number of base vectors, i.e., 19.
4. Vectors are mapped by taking their position in the list above and setting the component on that index to 1, for all others on 0. We use transposition depict the vectors in a linear fashion.

v_{this}	$(1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)^T$
v_{is}	$(0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)^T$
v_a	$(0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)^T$
$\vdots$	$\vdots$
$v_{webpages}$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)^T$

5. The documents are the sums of their components in the appropriate base vectors' directions.

$$\begin{aligned} D_1 &= \text{this is a text about web science} \\ &= tf(this, D_1)v_{this} + ... + tf(webpages, D_1)v_{webpages} \\ &= (1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0)^T \end{aligned}$$

$$\begin{aligned} D_2 &= \text{web science is covering the analysis of text corpora} \\ &= tf(this, D_2)v_{this} + ... + tf(webpages, D_2)v_{webpages} \\ &= (0,1,0,1,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0)^T \end{aligned}$$

$$\begin{aligned} D_3 &= \text{scientific methods are used to analyze webpages} \\ &= tf(this, D_3)v_{this} + ... + tf(webpages, D_3)v_{webpages} \\ &= (0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1)^T \end{aligned}$$

6. The cosine similarity between pairs (D_1, D_3) as well as (D_2, D_3) is trivial: the respective vectors have no common terms, at least one of the vectors is zero for a given component (or term frequency). Therefore, the dot product is 0 and as a result, the cosine similarity is 0 as well ($\Theta = 90°$).

   For pair (D_1, D_2), the cosine simiarity is given by the following equation:

$$cos(\Theta) = \frac{< D_1, D_2 >}{||D_1|| \cdot ||D_2||} = \frac{4}{\sqrt{7} \cdot 3} \approx 0.503952631(\Theta \approx 59.738149692°)$$

7. Documents D_1 and D_2 are the most similar documents w.r.t. the cosine similarity.

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 D_1 and D_2 are closest to each other, as they share the most words. They share $\{is, text, web, science\}$ where the latter shares none. Some problems are: concatenated words ("webpages" vs "web pages"), unstemmed words ("scientific" and "science" having a close meaning but not being lexically equivalent) and common english words ("is", "are", "the"). Those problems are usually solved by stemming (e.g. Porter stemming) and stop-word removal.

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.

Answers 2.1 build a generator

Listing 1 Task 2.1 build a generator

```

# coding: utf-8

from collections import Counter
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
import sortedcontainers as c
import random

def convert_dict(d):
    s = pd.Series(d)
    s.index.name='Char'
    s.reset_index()
    df = pd.DataFrame({'Char':s.index, 'P':s.values})
    df['S'] = np.cumsum(df['P'])
    return df

def generate_chars(file, distribution, n):
    d = convert_dict(distribution)
    sd = c.SortedDict((key, value) for key, value in zip(d['S'],d[
        ↪ 'Char']))
    str=[]
    for i in range(1,n+1):
        index = sd.bisect(random.random())
        key = sd.iloc[index]
        str.append(sd[key])
        if (i % 1000000 == 0 or i==n):
            text=''.join(str)
            print(i)
            append_to_file(file, text)
            str=[]

# Create a new file
def write_file(file):
    with open(file, 'w') as f:
        f.write('')

# Add data onto an existing file
def append_to_file(file, data):
    with open(file, 'a') as file:
        file.write(data)



```

```
if __name__ == '__main__':

    zipf_probabilities = {' ': 0.17840450037213465, '1':
        ↪ 0.004478392057619917, '0': 0.003671824660673643, '3':
        ↪ 0.0011831834225755678, '2': 0.0026051307175779174, '5':
        ↪ 0.0011916662329062454, '4': 0.0011108979455528355, '7':
        ↪ 0.001079651630435706, '6': 0.0010859164582487295, '9':
        ↪ 0.0026071152282516083, '8': 0.0012921888323905763, '_':
        ↪ 2.3580656240324293e-05, 'a': 0.07264712490903191, 'c':
        ↪ 0.02563767289222365, 'b': 0.013368688579962115, 'e':
        ↪ 0.09688273452496411, 'd': 0.029857183586861923, 'g':
        ↪ 0.015076820473031856, 'f': 0.017232219565845877, 'i':
        ↪ 0.06007894642873102, 'h': 0.03934894249122837, 'k':
        ↪ 0.006067466280926215, 'j': 0.0018537015877810488, 'm':
        ↪ 0.022165129421030945, 'l': 0.03389465109649857, 'o':
        ↪ 0.05792847618595622, 'n': 0.058519445305660105, 'q':
        ↪ 0.0006185966212395744, 'p': 0.016245321110753712, 's':
        ↪ 0.055506530071283755, 'r': 0.05221605572640867, 'u':
        ↪ 0.020582942617121572, 't': 0.06805204881206219, 'w':
        ↪ 0.013964469813783246, 'v': 0.007927199224676324, 'y':
        ↪ 0.013084644140464391, 'x': 0.0014600810295164054, 'z':
        ↪ 0.0010488592883485 06}
    uniform_probabilities = {' ': 0.1875, 'a': 0.03125, 'c':
        ↪ 0.03125, 'b': 0.03125, 'e': 0.03125, 'd': 0.03125, 'g':
        ↪ 0.03125, 'f': 0.03125, 'i': 0.03125, 'h': 0.03125, 'k':
        ↪ 0.03125, 'j': 0.03125, 'm': 0.03125, 'l': 0.03125, 'o':
        ↪ 0.03125, 'n': 0.03125, 'q': 0.03125, 'p': 0.03125, 's':
        ↪ 0.03125, 'r': 0.03125, 'u': 0.03125, 't': 0.03125, 'w':
        ↪ 0.03125, 'v': 0.03125, 'y': 0.03125, 'x': 0.03125, 'z':
        ↪ 0.03125}

    file=r"simple-20160801-1-article-per-line"
    zipf_file=r"zipf2.txt"
    uniform_file=r"uniform2.txt"

    text = open(file, 'r', encoding='utf-8').read()
    n = len(text)

    write_file(zipf_file)
    write_file(uniform_file)

    generate_chars(zipf_file,zipf_probabilities, n)
    generate_chars(uniform_file,uniform_probabilities, n)
```

2.2 Plot the word rank frequency diagram and CDF

Listing 2 Task 2.2 Plot the word rank frequency diagram and CDF

```
from collections import Counter
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os

def readWords(file):
    f = open(file, 'r', encoding='utf-8')
    allWords = []
    for line in f:
        line = line[:-1]
        words = line.split()
        allWords.extend(words)
    return allWords

if __name__ == '__main__':

    file=r"simple-20160801-1-article-per-line"
    zipf_file=r"zipf2.txt"
    uniform_file=r"uniform2.txt"

    words_original = readWords(file)
    words_zipf = readWords(zipf_file)
    words_uniform = readWords(uniform_file)

    c_original = Counter(words_original)
    c_zipf = Counter(words_zipf)
    c_uniform = Counter(words_uniform)
    words_o, frequencies_o = zip(*c_original.most_common())
    words_z, frequencies_z = zip(*c_zipf.most_common())
    words_u, frequencies_u = zip(*c_uniform.most_common())

    cumsum_o = np.cumsum(frequencies_o)
    normedcumsum_o = [x / float(cumsum_o[-1]) for x in cumsum_o]
    # wrank_o = {words_o[i]:i+1 for i in range(0,len(words_o))}

    cumsum_z = np.cumsum(frequencies_z)
    normedcumsum_z = [x / float(cumsum_z[-1]) for x in cumsum_z]
    # wrank_z = {words_z[i]:i+1 for i in range(0,len(words_z))}

    cumsum_u = np.cumsum(frequencies_u)
    normedcumsum_u = [x / float(cumsum_u[-1]) for x in cumsum_u]
    # wrank_u = {words_u[i]:i+1 for i in range(0,len(words_u))}

    rank_o = [i + 1 for i in range(0, len(words_o))]
    rank_z = [i + 1 for i in range(0, len(words_z))]
    rank_u = [i + 1 for i in range(0, len(words_u))]

    # Word rank - Frequency
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.plot(rank_o, frequencies_o, label='original')
    ax.plot(rank_z, frequencies_z, label='generated zipf')
    ax.plot(rank_u, frequencies_u, label='generated uniform')

    # Now add the legend with some customizations.
    legend = ax.legend(loc='upper right', shadow=True)
    ax.set_yscale("log")
    ax.set_xscale("log")
    ax.set_ylabel(r"Frequency", fontsize=16, color="blue")
    ax.set_xlabel(r"Word rank", fontsize=16, color="blue")

    plt.show()

    # CDF
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.plot(rank_o, normedcumsum_o, label='original')
    ax.plot(rank_z, normedcumsum_z, label='generated zipf')
    ax.plot(rank_u, normedcumsum_u, label='generated uniform')

    # Now add the legend with some customizations.
    legend = ax.legend(loc='lower right', shadow=True)
    ax.set_yscale("log")
    ax.set_xscale("log")
    ax.set_ylabel(r"CDF", fontsize=16, color="blue")
    ax.set_xlabel(r"Word rank", fontsize=16, color="blue")

    plt.show()

    # Max point-wise distance
    d_zipf = max(map(lambda p: abs(p[0] - p[1]), (zip(
        ↪ normedcumsum_o, normedcumsum_z))))
    d_uniform = max(map(lambda p: abs(p[0] - p[1]), (zip(
        ↪ normedcumsum_o, normedcumsum_u))))

    print('Zipf:', d_zipf)
    print('\nUniform', d_uniform)
```

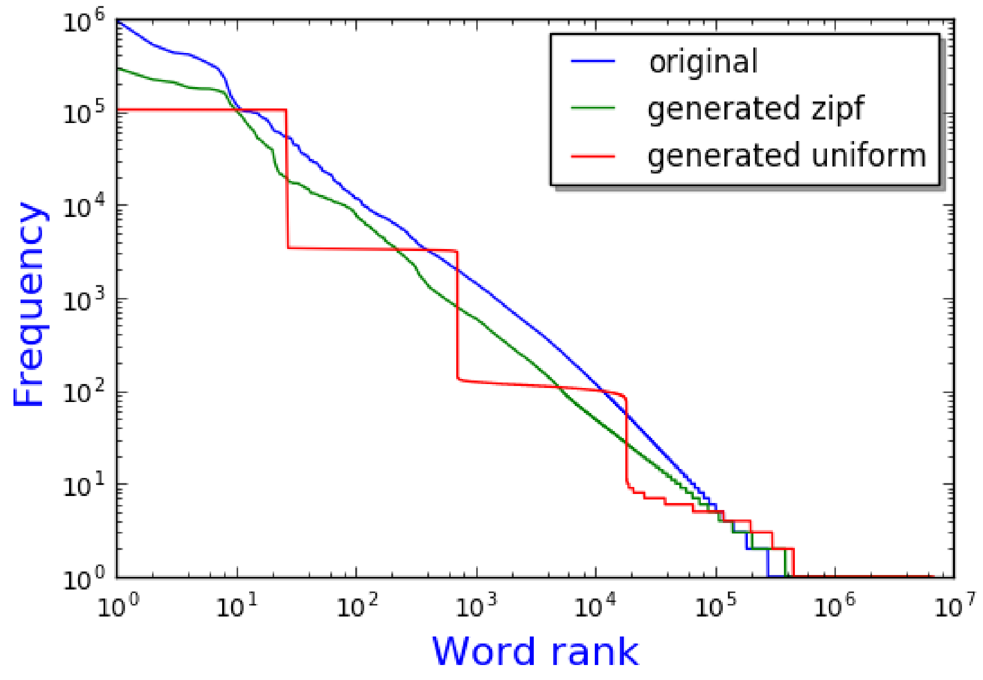


Figure 1: Word rank frequency

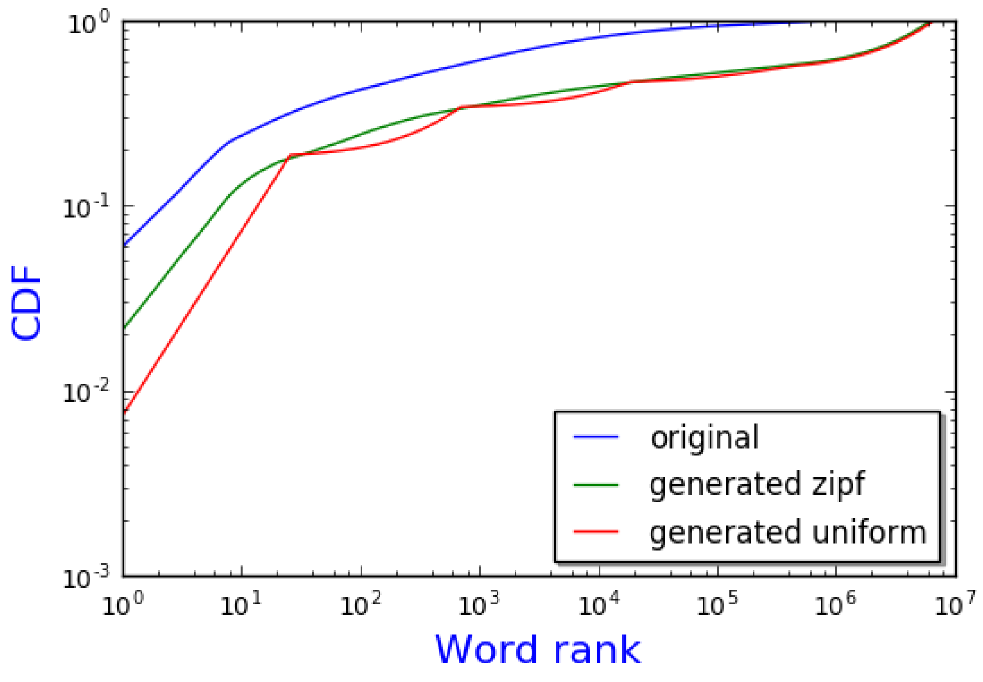
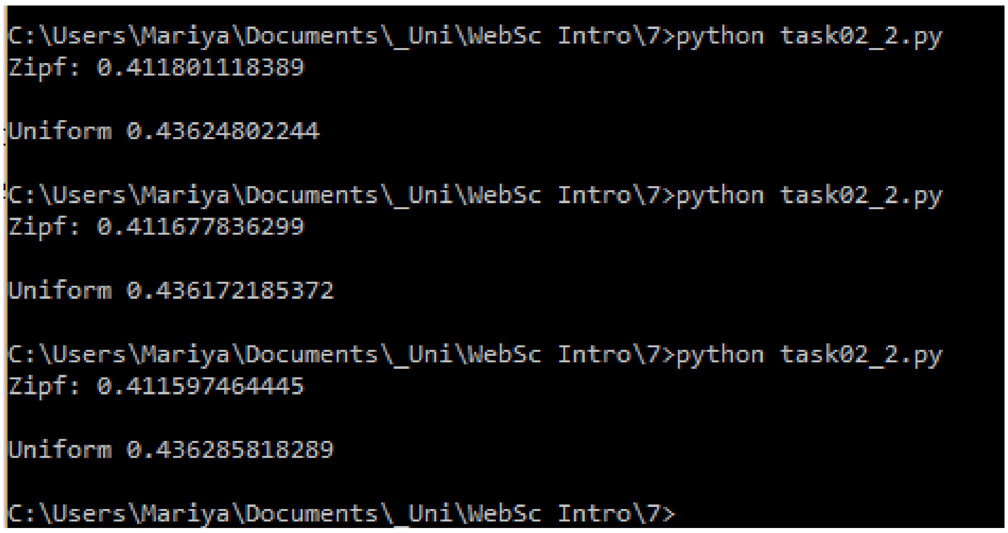


Figure 2: CDF



```
C:\Users\Mariya\Documents\_Uni\WebSc Intro\7>python task02_2.py
Zipf: 0.411801118389

Uniform 0.43624802244

C:\Users\Mariya\Documents\_Uni\WebSc Intro\7>python task02_2.py
Zipf: 0.411677836299

Uniform 0.436172185372

C:\Users\Mariya\Documents\_Uni\WebSc Intro\7>python task02_2.py
Zipf: 0.411597464445

Uniform 0.436285818289

C:\Users\Mariya\Documents\_Uni\WebSc Intro\7>
```

Figure 3: Console

2.3 Comparison

Generator that used zipf distribution performs better as it has the minimum point-wise distance compared with uniform distibution generator. This was confirmed by 3 tests conducted:

Max Pointwise distances for 3 tests:

Test 1

Zipf: 0.411801118389 Uniform: 0.43624802244

Test 2

Zipf: 0.411677836299 Uniform: 0.436172185372

Test 3

Zipf: 0.411597464445 Uniform: 0.436285818289

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.

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)?

1. See: Figure 4
2. See: Figure 5
3.
   - Median: 7.0
   - $P(X \leq 9) : 0.77$
4. maximum point wise distance: 0.07

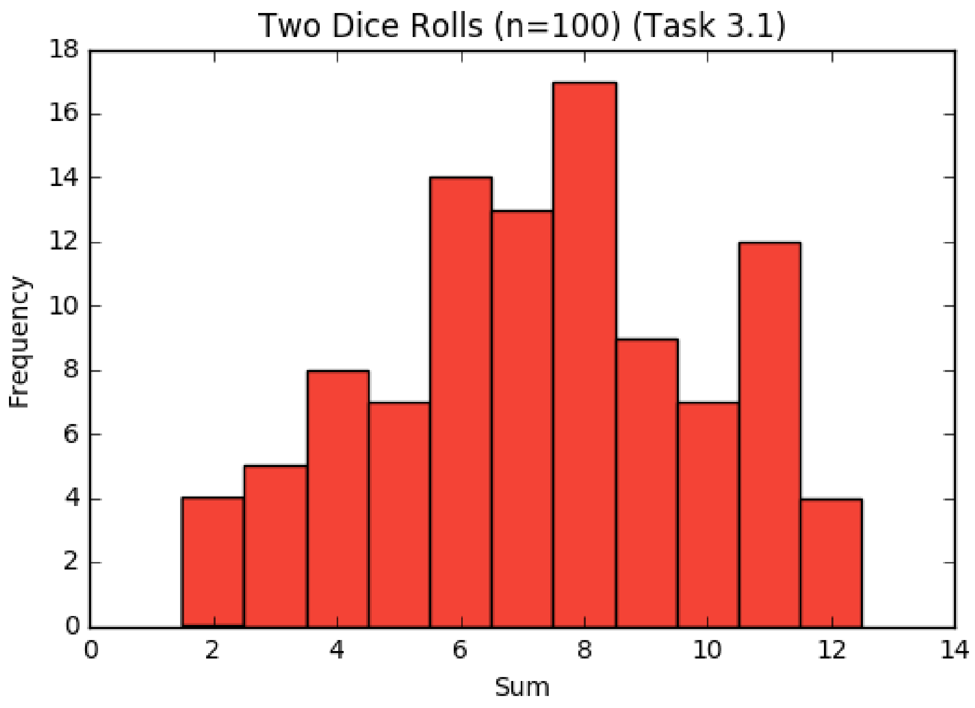


Figure 4: Task 3.1

5. maximum point wise distance: 0.029
6. The maximum point wise distance is less than a half of the maximum point wise distance than with n = 100. But increasing only increases the probability of having a smaller maximum point wise distance.

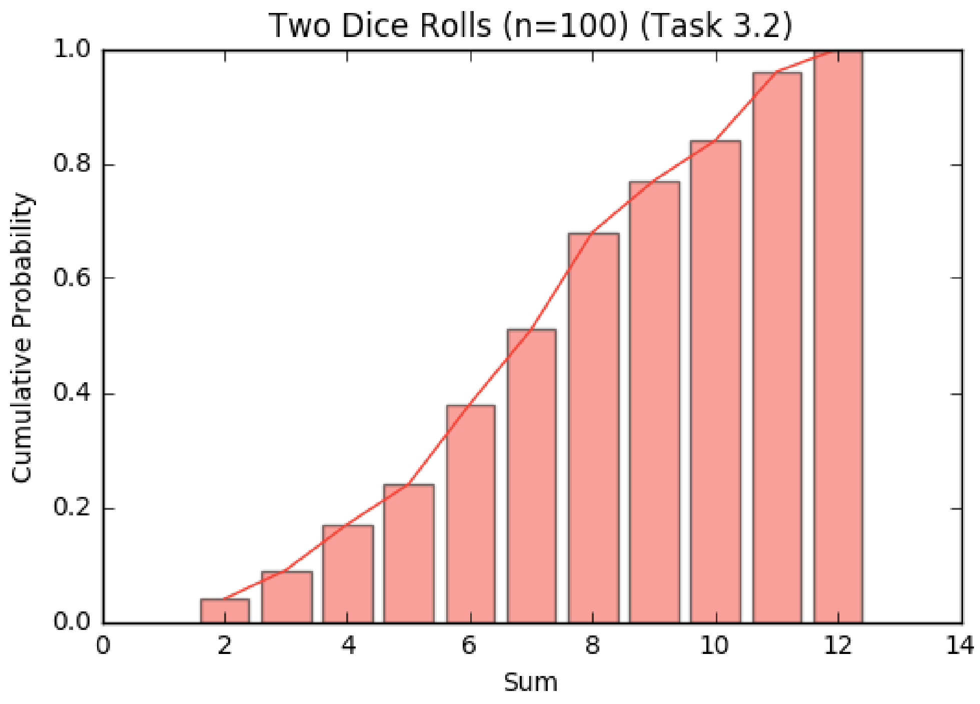


Figure 5: Task 3.2

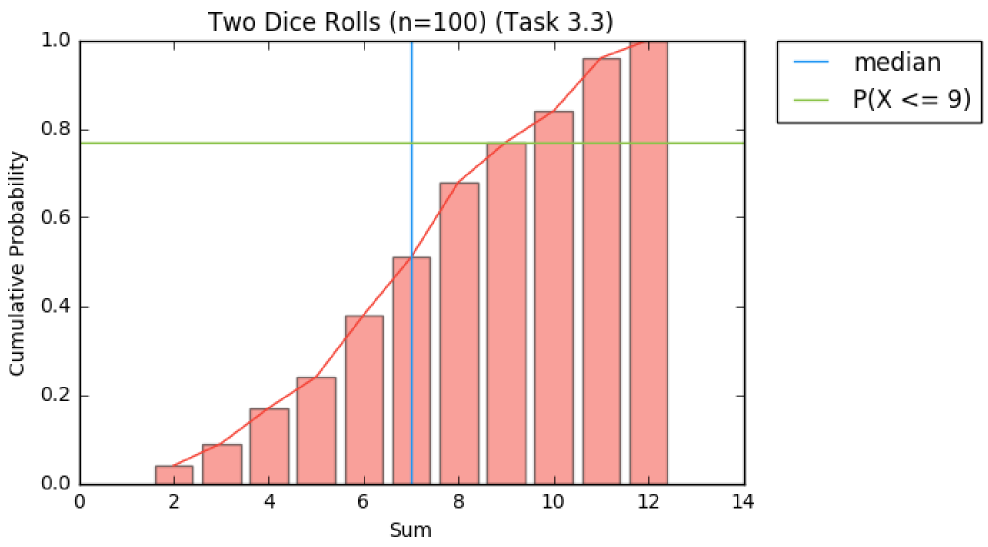


Figure 6: Task 3.3

Listing 3 Task 3 python code

```python
import concurrent.futures
import matplotlib.pyplot as plt
import numpy as np
import random
from collections import Counter

def repeat(n, fn):
    return list(map(lambda _: fn(), range(0, n)))


def cdf(lst):
    numbers, frequencies = zip(*sorted(Counter(lst).items()))
    cumsum = np.cumsum(frequencies)
    normed_cumsum = [x / float(cumsum[-1]) for x in cumsum]
    return numbers, normed_cumsum


class DiceModel:
    def __init__(self, sides=6):
        self.sides = sides

    def roll(self):
        return random.randint(1, self.sides)


class DiceSumModel:
    def __init__(self, rolls, dices):
        self.rolls = rolls
        self.dices = [DiceModel() for _ in range(0, dices)]

    def evaluate(self):
        results = {}
        with concurrent.futures.ThreadPoolExecutor() as executor:
            future_to_dice = {executor.submit(repeat, self.rolls,
                ↪ dice.roll): dice for dice in self.dices}
            for future in concurrent.futures.as_completed(
                ↪ future_to_dice):
                dice = future_to_dice[future]
                result = future.result()
                results[dice] = result
        return list(map(sum, (zip(*results.values()))))

def task_3_1():
    sums = DiceSumModel(100, 2).evaluate()
    plt.hist(sums, bins=range(2, 14), align='left', color='#F44336
        ↪ ')
    plt.title('Two Dice Rolls (n=100) (Task 3.1)')
    plt.xlabel('Sum')
    plt.ylabel('Frequency')
    plt.show()
    return sums


def task_3_2(task_3_1_sums):
    task_3_2_eyes, task_3_2_cdf = cdf(task_3_1_sums)
    plt.bar(task_3_2_eyes, task_3_2_cdf, align='center', alpha
        ↪ =0.5, color='#F44336')
    plt.plot(task_3_2_eyes, task_3_2_cdf, color='#F44336')
    plt.title('Two Dice Rolls (n=100) (Task 3.2)')
    plt.xlabel('Sum')
    plt.ylabel('Cumulative Probability')
    plt.show()


def task_3_3(task_3_1_sums):
    task_3_3_eyes, task_3_3_cdf = cdf(task_3_1_sums)
    median = np.median(task_3_1_sums)
    leq_9_prob = task_3_3_cdf[task_3_3_eyes.index(9)]
    plt.bar(task_3_3_eyes, task_3_3_cdf, align='center', alpha
        ↪ =0.5, color='#F44336')
    plt.plot(task_3_3_eyes, task_3_3_cdf, color='#F44336')
    plt.axvline(x=median, c='#2196F3', label='median', lw=1)
    plt.axhline(y=leq_9_prob, c='#8BC34A', label='P(X <= 9)', lw
        ↪ =1)
    plt.title('Two Dice Rolls (n=100) (Task 3.3)')
    plt.xlabel('Sum')
    plt.ylabel('Cumulative Probability')
    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
    print('Task 3.3 median: %s' % median)
    print('Task 3.3 P(X <= 9): %s' % leq_9_prob)
    plt.show()


def task_3_4(task_3_1_sums):
    task_3_4_sums = DiceSumModel(100, 2).evaluate()
    task_3_1_eyes, task_3_1_cdf = cdf(task_3_1_sums)
    task_3_4_eyes, task_3_4_cdf = cdf(task_3_4_sums)
    task_3_4_max_pwd = max(map(lambda p: abs(p[0] - p[1]), (zip(
        ↪ task_3_1_cdf, task_3_4_cdf))))
    print('Task 3.4 maximum point wise distance: %s' %
        ↪ task_3_4_max_pwd)


def task_3_5():
    task_3_5_1_sums = DiceSumModel(1000, 2).evaluate()
    task_3_5_2_sums = DiceSumModel(1000, 2).evaluate()
    task_3_5_1_eyes, task_3_5_1_cdf = cdf(task_3_5_1_sums)
    task_3_5_2_eyes, task_3_5_2_cdf = cdf(task_3_5_2_sums)
    task_3_4_max_pwd = max(map(lambda p: abs(p[0] - p[1]), (zip(
        ↪ task_3_5_1_cdf, task_3_5_2_cdf))))
    print('Task 3.5 maximum point wise distance: %s' %
        ↪ task_3_4_max_pwd)


if __name__ == "__main__":
    task_3_1_sums = task_3_1()
    task_3_2(task_3_1_sums)
    task_3_3(task_3_1_sums)
    task_3_4(task_3_1_sums)
    task_3_5()
```

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`.