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: uniform
Group members: Pradip Giri, Jalak Arvind Kumar Pansuriya, Madhu Rakhal Magar

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.

Answers:
1.1
Without applying any modeling methods we can say that two documents are similar if they contain some of the same term. The possible measures of similarity might be:
The length of the document, the number of terms in common, whether the terms are common or unusual and how many times the each term appears etc. While considering above facts we can say that the document D1 is much more similar to document D2. There is some common terms in document D1 and D2. The semantic of the D1 and D2 look same, both are somehow talking about web science.

1.2

1. In the given three documents there are 19 different words(terms) so, 19 base vectors would be needed to model the documents of this corpus.
2. Each dimensions of the vector space stands for term/token used to index a set of documents.
3. The vector space have 19 dimensions.
4. A table to map the words of the documents to the base vectors:

   In our documents D_1, D_2, and D_3 we have 19 different words.
   Let V = <$\vec{T}_1$, $\vec{T}_2$ .... , $\vec{T}_{19}$ > be the vector space spanned by the words “this”, ”is”, ”a”, ”text”...... (all the different words in three documents.)

$$\vec{T}_1 = \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, \vec{T}_2 = \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, \vec{T}_3 = \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, ..., \vec{T}_{19} = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$$

and so on...

Now, in every document there is no repetition of any words. That means the term frequency of each words in a document is = 1. The below table shows the Weighting by Term Frequency (tf):

-	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
d_1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
d_2	0	1	0	1	0	1	1	1	1	1	1	1	0	0	0
d_3	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

-	T_{16}	T_{17}	T_{18}	T_{19}	length
d_1	0	0	0	0	$\sqrt(7)$
d_2	0	0	0	0	3
d_3	1	1	1	1	$\sqrt(7)$

where T means Term in document vector space.

5. The document vector of a document D_1 is:

$$\vec{d_1} = \sum_{i=1}^{7} tf(w_i, D_1)w_i \tag{1}$$

$$= tf(T_1, D_1)\vec{T_1} + tf(T_2, D_1)\vec{T_2} + tf(T_3, D_1)\vec{T_3} + tf(T_4, D_1)\vec{T_4} + tf(T_5, D_1)\vec{T_5} + tf(T_6, D_1)\vec{T_6} + tf(T_7, D_1) \tag{2}$$

$$= \quad 1 * \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + 1 * \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + ... + 1 * \begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

$=$

$$\vec{d_1} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Similarly, the document vector for D_2 and D_3 is given by:

$$\vec{d_2} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad and \quad \vec{d_3} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

6. The cosine similarity between two documents are given by:

$$cos(\theta) = \frac{d_1.d_2}{|d_1|.|d_2|} \tag{3}$$

	d1	d2	d3
d1	1	0.50	0
d2	0.50	1	0
d3	0	0	1

$cos(\theta) = 1$; documents exactly the same; $= 0$, totally different
From the above table we can see that the cosine similarity between d1 and d2 is = 0.50
The cosine similarity between d1 and d3 = 0.
The cosine similarity between d2 and d3 = 0

7. According to the cosine similarity document d1 document d2 are most similar.

1.3
The results of the model and our expectation from sub-task1.1 is matched. Each dimensions of the vector space is represented by each term/token in the documents where term/token refer to the unique word in the documents. As we know the cosine similarity of two documents is give by $cos(\theta)$ as in equation1. If the number of similar words in two document are higher then the value of$cos(\theta)$ is also high. Because of this our expectation matched.

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?

Answer 2.1

Listing 1 helper.py

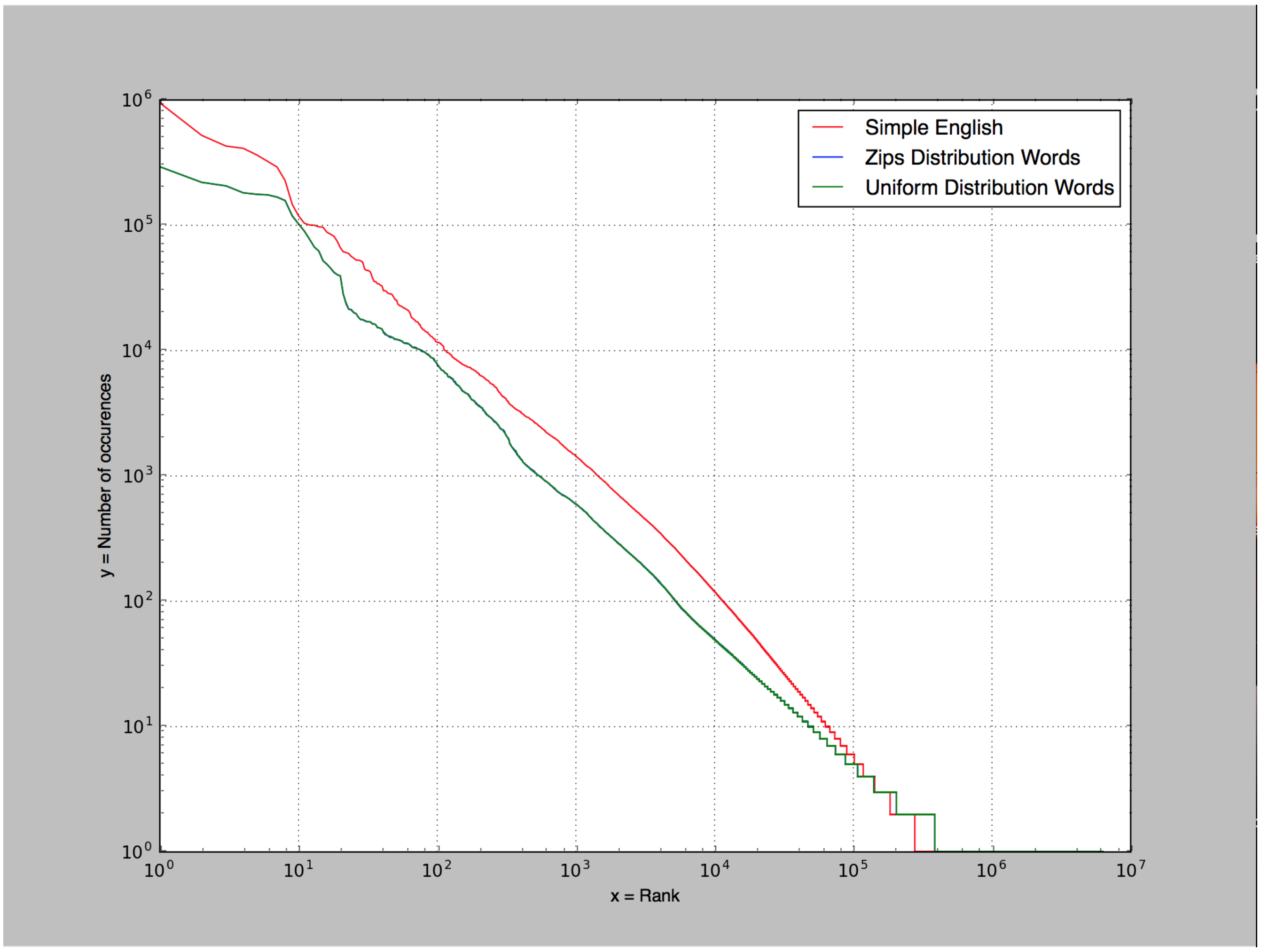
```
"""
 Helper Methods
"""
# pylint: disable-msg=C0103
from threading import Thread
import random
import bisect
from collections import Counter
from functools import reduce
from copy import deepcopy
import numpy as np
import matplotlib.pyplot as plt

THREADS = 30

def remove_special_char(regex, string):
    """removes any character than a-z and 0-9"""
    return regex.sub('', string.lower())


def read_file_content(file_path):
    """ open given file, read and return the content
    None will be returned if error while opening the file
    """
    content = None
    with open(file_path, 'r+') as f:
        content = f.read()
    return content


def chunks(lis, num):
    """ returns the chunks """
    n = max(1, num)
    return (lis[i:i+n] for i in range(0, len(lis), n))


def save_file(file_name, content):
    """ writes to the file"""
    with open(file_name, 'w+') as f:
        f.write(content)
    print("{} written sucessfully".format(file_name))


def cal_probability(list_of_touple):
    """ calculates the probability each key from given list"""
    total = reduce(lambda x, y: x + y[1], list_of_touple, 0)
    return[{x[0]: x[1]/total} for x in list_of_touple]


def cal_cum_frequency(input_list_of_dict):
    """ calculates the cumulative frequency"""
    list_of_dict = deepcopy(input_list_of_dict)
    for index, obj in enumerate(list_of_dict, start=1):
        req_dicts = list_of_dict[:index]
        total = 0
        for req_dict in req_dicts:
            for key in req_dict.keys():
                if key != 'cum_freq':
                    total += req_dict[key]
        obj['cum_freq'] = total
    return list_of_dict


def _generate_text(char_count, keys, keys_cdfs, result, index):
    """ generate text using thread"""
    characters = []
    for x in range(0, char_count // THREADS):
        randomValue = random.random()
        idx = bisect.bisect(keys_cdfs, randomValue)
        characters.append(keys[idx])
    text = "".join(characters)
    result[index] = text
    return text

def gen_random_text(char_count, keys, keys_cdfs):
    """ generate random text"""
    threads = [None] * 1000
    results = {}
    i = 0
    for i in range(0, len(threads)):
        threads[i] = Thread(target=_generate_text,\
                                            args=(char_count, keys, keys_cdfs, results, i
        threads[i].start()

    for thread in threads:
        thread.join()

    return ''.join(list(results.values()))


def get_word_with_probability(text):
    """retun words with its probability"""
    freq_each_words = Counter(text.split()).most_common()
    sum_of_total_char_occ = sum([frequency for (key, frequency) in freq_each_words
    prob_each_word = {}
    for (key, frequency) in freq_each_words:
        prob_each_word[key] = frequency / sum_of_total_char_occ
    return prob_each_word


def draw_plot(listElements_S, listElements_Z, listElements_U, xLabel, yLabel):
    """ draws the plot"""
    x_S = [x for x in range(1, len(listElements_S)+1)]
    y_S = np.array(listElements_S)
    x_Z = [x for x in range(1, len(listElements_Z)+1)]
    y_Z = np.array(listElements_Z)
    x_U = [x for x in range(1, len(listElements_U)+1)]
    y_U = np.array(listElements_U)
    plt.figure(figsize=(12, 9))
    plt.plot(x_S, y_S, 'r', label="Simple English")
    plt.plot(x_Z, y_Z, 'b', label="Zips Distribution Words")
    plt.plot(x_U, y_U, 'g', label="Uniform Distribution Words")
    plt.xlabel(xLabel)
    plt.ylabel(yLabel)
    plt.yscale('log')
    plt.xscale('log')
    plt.legend(loc='upper right')
    plt.grid()
    plt.show()


def draw_cdf_plot(listElements_S, listElements_Z, listElements_U, xLabel, yLabel)
    """draws the cdf plot"""
    x_S = [x for x in range(1, len(listElements_S) + 1)]
    y_S = np.array(listElements_S)
    x_Z = [x for x in range(1, len(listElements_Z) + 1)]
    y_Z = np.array(listElements_Z)
    x_U = [x for x in range(1, len(listElements_U) + 1)]
    y_U = np.array(listElements_U)
    plt.figure(figsize=(12, 9))
    plt.plot(x_S, y_S, 'r', label="Simple English")
    plt.plot(x_Z, y_Z, 'b', label="Zips Distribution Words")
    plt.plot(x_U, y_U, 'g', label="Uniform Distribution Words")
    plt.xlabel(xLabel)
    plt.ylabel(yLabel)
    plt.xscale('log')
    plt.yscale('log')
    plt.ylim(0, 1.2)
    plt.legend(loc='bottom right')
    plt.grid()
    plt.show()
```

Listing 2 assignment7.py

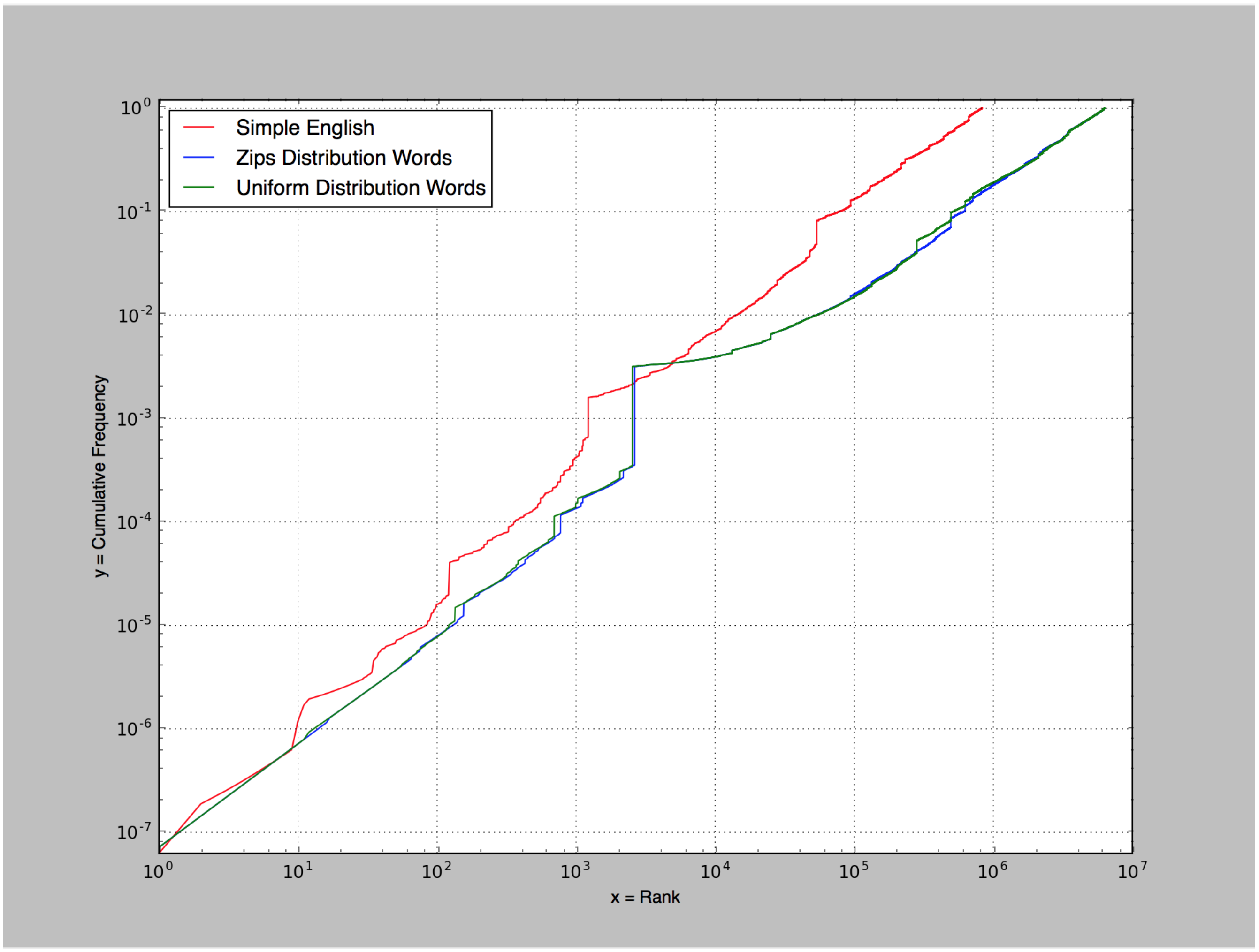
```
"""
 Generative Model
"""
# pylint: disable-msg=C0103
import re
import threading
from timeit import default_timer as timer
from pathlib import Path
from collections import Counter
from operator import itemgetter
import numpy as np
from helper import read_file_content
```

```python
from helper import chunks
from helper import remove_special_char
from helper import save_file
from helper import cal_probability
from helper import cal_cum_frequency
from helper import draw_plot
from helper import gen_random_text
from helper import get_word_with_probability
from helper import draw_cdf_plot


counter = Counter()
total_character_in_english_text = 0
probabilities_file = 'probabilities.py.txt'
regex = re.compile('[^a-zA-Z0-9 ]')

def cdf_calculaton(prob_dict):
    """ calculates cdf"""
    arr_for_cdf = list(prob_dict.values())
    a = np.array(arr_for_cdf)
    cdfEvalDict = np.cumsum(a)
    return (list(prob_dict.keys()), cdfEvalDict)

def count_alphanumberic(string):
    """ count frequency of each alphabet"""
    global counter
    clean_string = remove_special_char(regex, string)
    ctr = Counter(clean_string)
    counter = counter + ctr

def perform_kolmogro_test(cdf_main , cdf_compare):
    """ perform kolmogro test"""
    max_point_wise_distance = max([abs(cdf_S - cdf_Zipf) \
                                   for (cdf_S, cdf_Zipf) in \
                                               zip(cdf_main, cdf_compare)])
    return max_point_wise_distance

def process_articles(articles):
    """ access all the article from articles"""
    for article in articles:
        count_alphanumberic(article)


def use_multithread(articles):
    """"" using multi threading"""
    group_articles = chunks(articles, int(len(articles) / 20))
    thread_list = []
    for art in group_articles:
        t = threading.Thread(target=process_articles, args=(art, ))
        thread_list.append(t)
        t.start()

    for thread in thread_list:
        thread.join()


def without_multi(articles):
    """ processing in single thread"""
    for article in articles:
        count_alphanumeric(article)


def process_output_file(file_path):
    """ read already generated output and process it"""
    global counter
    output = None
    with file_path.open() as f:
        output = f.read()
    counter = eval(output)


def load_file_and_process(file_path):
    """ load the simple article file and process it"""
    global counter
    content = read_file_content(file_path)
    start = timer()
    articles = content.split("\n")
    use_multithread(articles)
    end = timer()
    print("Total time taken:", end - start)
    # sorting the counter
    counter = sorted(counter.items(), key=itemgetter(0))
    save_file('output.txt', str(counter))


def process_given_info_file(file_path):
    """ process the given file which contains probability information"""
    def get_text(text, start, end):
        """ returns slice of text from stat to end:"""
        return text[text.find(start): text.find(end) + 1]

    def process_list(given_list):
        """ returns dict"""
        return list([{x[0]: x[1]} for x in given_list])
    file_content = read_file_content(file_path)
    text_arr = file_content.split("\n")
    texts = list(filter(lambda x: len(x.strip()) > 0, text_arr))
    result = list(map(lambda x: get_text(x, '{', '}'), texts))
    final = []
    for res in result:
        jt = eval(res)
        mp = sorted(jt.items(), key=itemgetter(0))
        final.append(mp)
    return list(map(lambda x: process_list(x), final))



def get_keys_and_cdfs(list_of_dict):
    """ returns the keys and its cumulative frequency"""
    cums = []
    keys = []
    for cur_dict in list_of_dict:
        for key in cur_dict.keys():
            if key == 'cum_freq':
                cums.append(cur_dict[key])
            else:
                keys.append(key)
    return [keys, cums]


def main(file_path):
    """ entry point of the application"""
    global total_character_in_english_text
    simple_english_text = read_file_content(file_path)
    total_character_in_english_text = len(simple_english_text)
    prob_each_word_dict = get_word_with_probability(simple_english_text)
    _, assoc_cdf_s = cdf_calculaton(prob_each_word_dict)

    # for Zipf and uniform distrubation
    prob_dist_given = process_given_info_file(probabilities_file)
    zipf_cum = cal_cum_frequency(prob_dist_given[0])
    unif_cum = cal_cum_frequency(prob_dist_given[0])
    all_keys_zipf, asso_zipf_cum = get_keys_and_cdfs(zipf_cum)
    all_keys_unif, asso_unif_cum = get_keys_and_cdfs(unif_cum)

    text_zipf = gen_random_text(total_character_in_english_text, all_keys_zipf, a
    save_file('text_zipf.txt', text_zipf)

    text_unif = gen_random_text(total_character_in_english_text, all_keys_unif, a
    save_file('text_unif.txt', text_unif)

    prob_for_each_ord_dict_zipf = get_word_with_probability(text_zipf)
    _, ass_cdf_new_zipf = cdf_calculaton(prob_for_each_ord_dict_zipf)

    prob_for_each_word_dict_unif = get_word_with_probability(text_unif)
    _, asso_cdf_new_unif = cdf_calculaton(prob_for_each_word_dict_unif)
    #--------------------------------------------------------------------------
    # Plots for Rank Frequency STARTS
    #---------------------------------------------------------------------------
    #Note: _S is for simple English
    freq_each_word_s = Counter(simple_english_text.split()).most_common()
    list_for_rank_freq_diag_s = [freq for (word, freq) in freq_each_word_s]

    #Note _Z is for Zipf
    freq_each_word_z = Counter(text_zipf.split()).most_common()
    list_for_rank_freq_diag_z = [freq for (word, freq) in freq_each_word_z]

    #Note_U is for Uniform
    freq_each_word_u = Counter(text_unif.split()).most_common()
    list_for_rank_freq_diag_u = [freq for (word, freq) in freq_each_word_u]

    draw_plot(list_for_rank_freq_diag_s, list_for_rank_freq_diag_z,\
             list_for_rank_freq_diag_u, "x = Rank", "y = Number of occurences")
    #---------------------------------------------------------------------------
    # Plots for Rank Frequency ENDS
    #---------------------------------------------------------------------------

    #---------------------------------------------------------------------------
    # Plots for Rank and CDF STARTS
    #---------------------------------------------------------------------------

    draw_cdf_plot(assoc_cdf_s, ass_cdf_new_zipf, asso_cdf_new_unif,\
     "x = Rank", "y = Cumulative Frequency")

    #---------------------------------------------------------------------------
    # Plots for Rank and CDF ENDS
    #---------------------------------------------------------------------------
    # Now we perform Kolmogorov Smirnov test  by calculating the maximum
    # pointwise distance of CDFs
    max_point_wised_s_zipf = perform_kolmogro_test(assoc_cdf_s, ass_cdf_new_zipf)
    max_point_wised_s_unif = perform_kolmogro_test(assoc_cdf_s, asso_cdf_new_unif
    print("Obtained Maximum Pointwise Distance between simple English \
                                   and Zipf is : ", max_point_wised_s_zipf)
    print("Obtained Maximum Pointwise Distance between simple English \
                                   and Unif is : ", max_point_wised_s_unif)

if __name__ == "__main__":
    file = 'simple-20160801-1-article-per-line'
    main(file)
```

Answer 2.2



Word frequencies depending on word



Cumulative word probability depending on word rank

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 o and be said inconsistent as we now observe that during the repetition of generating two text corpora our Pointwise distance di ered 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.

3.2 Only for nerds and board students (0 Points)

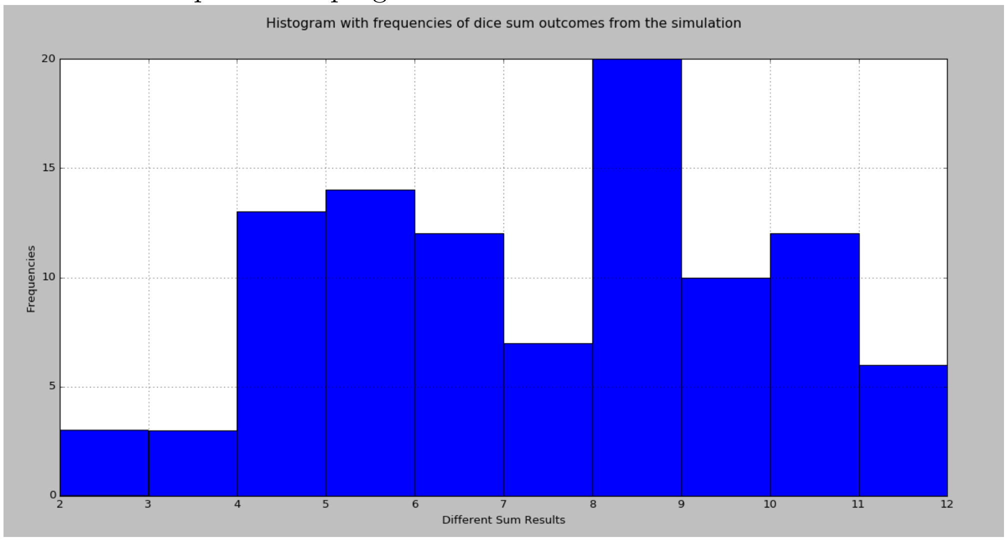
Assuming 20 groups of students. What is the likelihood that at least two groups come up with the same histograms in the case for n (=100)?

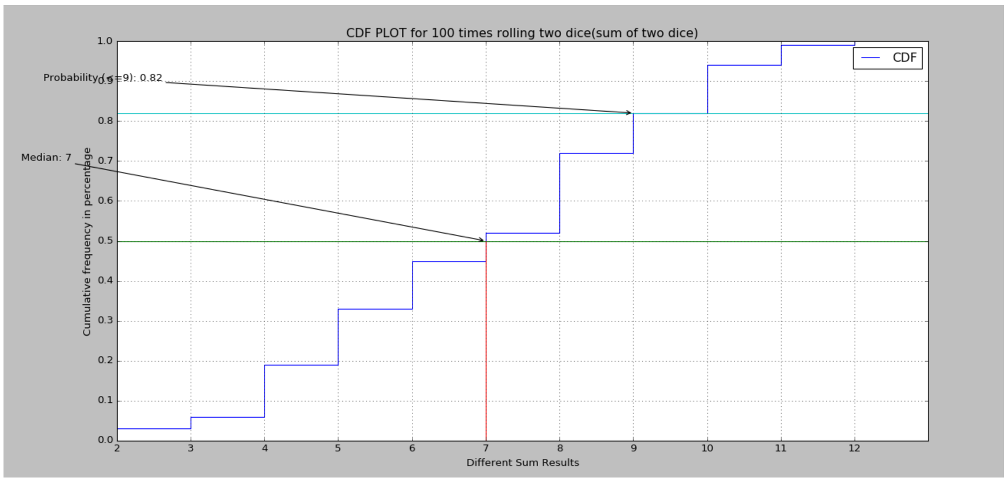
Answers:

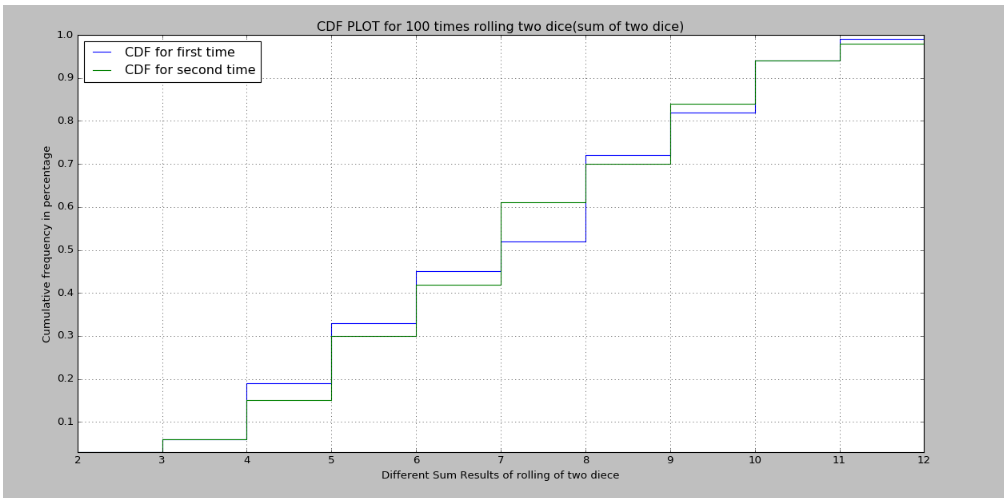
The required python program for above question is given below:

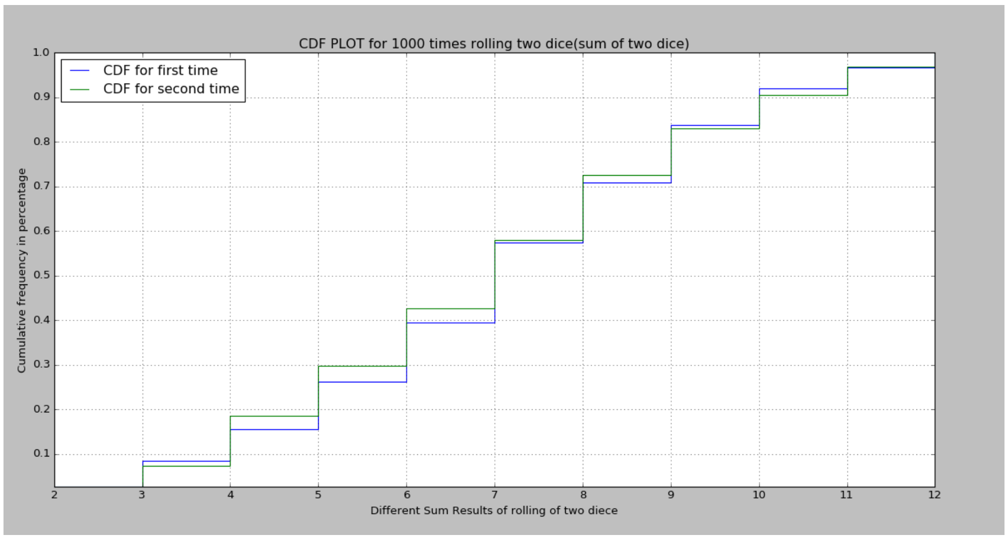
```
[language=Python, frame=single, caption=uniform_assignment7_Q3.py]
import random
import numpy as np
import matplotlib.pyplot as plt
import collections

#function to rolling two diece simultaneously and returns the sum of results
def rollDiceGetSum():
    return random.randint(1, 6) + random.randint(1, 6)

#functions to calculate CDF
def calculateCDF(result):
    frequencyNum = collections.Counter(result).most_common()
    total = sum([frequency for (key, frequency) in frequencyNum])
    probabilityForEachNumDict = {}
    for (key, frequency) in frequencyNum:
        probabilityForEachNumDict[key] = frequency / \
                                          total

    arrayForCDFCalc = list(probabilityForEachNumDict.values())
    a = np.array(arrayForCDFCalc)  # Gets us CDF
    cdfEvalDict = np.cumsum(a)
    return (list(probabilityForEachNumDict.keys()), cdfEvalDict)

# function to draw the histogram of sum of results
def drawHistogram(sumResult):
    plt.hist(sumResult, bins=[2,3,4,5,6,7,8,9,10,11,12])
    plt.title('Histogram with frequencies of dice sum outcomes from the simulatio
    plt.ylabel('Frequencies')
    plt.xlabel('Different Sum Results')
    plt.xticks(np.arange(2, 13, 1))
    plt.grid('on')
    plt.show()

#functions to draw CDF and shows the median and prob. of sum less or equal to 9
def drawSingleCDF(sumResult, median, probLessNnine):
    (xval, cdfval) = calculateCDF(sumResult)

    plt.title("CDF PLOT for 100 times rolling two dice(sum of two dice)")
    plt.xlabel("Different Sum Results")
    plt.ylabel("Cumulative frequency in percentage")
    plt.grid('on')
    plt.xticks(np.arange(2, 13, 1))
    plt.yticks(np.arange(0, 1.1, 0.1))
    plt.plot(xval, cdfval, drawstyle='steps-post', label='CDF')

    plt.plot((2, 13), (.5, .5))
    plt.plot((median, median), (0, .5))
    plt.annotate('Median: %d' % median, (median, .5), xytext=(0.7, 0.7), arrowpro

    plt.plot((2, 13), (probLessNnine, probLessNnine))
    plt.annotate('Probability (<=9): %s' % probLessNnine, (9, probLessNnine), xyt
                 arrowprops=dict(arrowstyle='->'))

    plt.legend(loc=0)
    plt.show()

#functions to draw two CDF according to random number n
def drawDoubleCDF(result1, result2, n):
    (xval1, cdfval1) = calculateCDF(result1)
    (xval2, cdfval2) = calculateCDF(result2)
    maxDist = getMaxDistance(cdfval1, cdfval2)

    print(" The maximum pointwise distance between two CDFs for n= " + str(n)+ ",

    plt.title("CDF PLOT for " + str(n) + " times rolling two dice(sum of two dice
    plt.xlabel("Different Sum Results of rolling of two diece")
    plt.ylabel("Cumulative frequency in percentage")
    plt.xticks(np.arange(2, 13, 1))
    plt.yticks(np.arange(0, 1.1, 0.1))

    plt.plot(xval1, cdfval1, drawstyle='steps-post', label='CDF for first time')
    plt.plot(xval2, cdfval2, drawstyle='steps-post', label='CDF for second time')

    plt.grid('on')
    plt.legend(loc=0)
    plt.show()

#functions to roll two dice uniformly upto n times.
def rollDice(n):
    sumResult = []
    for _ in range(n):
        result = rollDiceGetSum()
        sumResult.append(result)

    sumResult = np.sort(sumResult)
    return sumResult

#functions to calculate median from the give array
def getMedian(sumResult):
    a = np.array(sumResult)
    print("Median val: ", np.median(a))
    return np.median(a)

#functions to calculate probability of given value in a array
def getProbByVal(arr, val):
    gen = [x for x in arr if x<= val]
    return (len(gen)/len(arr))

#functions to calculate maximum pointwise distance
def getMaxDistance(result1, result2):
    diffResult = [abs(x - y) for x, y in zip(result1, result2)]
    return max(diffResult)

#main function to do random sampling according to question
def main():
    sumResult = rollDice(100)

    # Draw histogram
    drawHistogram(sumResult)

    median = getMedian(sumResult)
    val = getProbByVal(sumResult, 9)
    # Draw CDF
    drawSingleCDF(sumResult, median, val)

    sumResult1 = rollDice(100)

    # Draw CDF for n = 100
    drawDoubleCDF(sumResult, sumResult1, 100)

    # Draw CDF for n= 1000
    result1 = rollDice(1000)
    result2 = rollDice(1000)
    drawDoubleCDF(result1, result2, 1000)



if __name__ == '__main__':
    main()
```

Here is the output of the program:




CDF PLOT for 100 times rolling two dice(sum of two dice)
CDF
Probability(x<=9): 0.82
Median: 7
Cumulative frequency in percentage
Different Sum Results
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
2
3
4
5
6
7
8
9
10
11
12


CDF PLOT for 100 times rolling two dice(sum of two dice)
CDF for first time
CDF for second time
Cumulative frequency in percentage
Different Sum Results of rolling of two diece
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
2
3
4
5
6
7
8
9
10
11
12



```
Median val:  7.0
 The maximum pointwise distance between two CDFs for n= 100, is =0.09
 The maximum pointwise distance between two CDFs for n= 1000, is =0.035
```

3.6
While increasing the number of steps in the simulation we got smaller value of maximum point-wise distance. This means if the number of steps or no. of experiment is increased then we will get the more similar CDF.

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`.