# 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: QUEBEC

1. Daniel Kostic
2. Stefan Vujovic
3. Igor Fedotov

# 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 In our opinion, documents $D_1$ and $D_2$ are most similar, because both documents are covering the same topic "Web Science". Also, these two documents share more words in common.

1.2.1 The number of base vectors needed is 19, because that is the number of unique words in the documents mentioned above.

1.2.2 Each dimension of the vector space stands for one unique word in the documents.

1.2.3 The vector space has 19 dimensions, because it has 19 base vectors.

1.2.4 Mapping of words to base vectors:

| Num | Word | Vector |
|---|---|---|
| 1 | this | [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] |
| 2 | is | [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] |
| ... | ... | |
| 17 | webpages | [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1] |

1.2.5

n = 17
Words W = $w_1$,$w_2$..$w_n$
Vectors V = $\vec{w_1}$, $\vec{w_n}$... $\vec{w_n}$
Documents vector:

$$d_j = \sum_{i=1}^{n} tf(w_i, D_j)\vec{w}_i$$

1.2.6

Calculation of cosine has been done in python, and is attached to the end of this PDF as task1.

```
print(cosine(d1_dict, d2_dict))
print(cosine(d1_dict, d3_dict))
print(cosine(d3_dict, d2_dict))
```

```
0.5039526306789696
0.0
0.0
```

1. According to cosine similarity, documents $D_1$ and $D_2$ are the most similar, and other pairs of documents have 0 similarity. This could have been expected as there are no common words in documents $D_1$ and $D_3$ and $D_2$ and $D_3$. A flaw of this model is that it doesn't recognize similar words as : science and scientific

# 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?

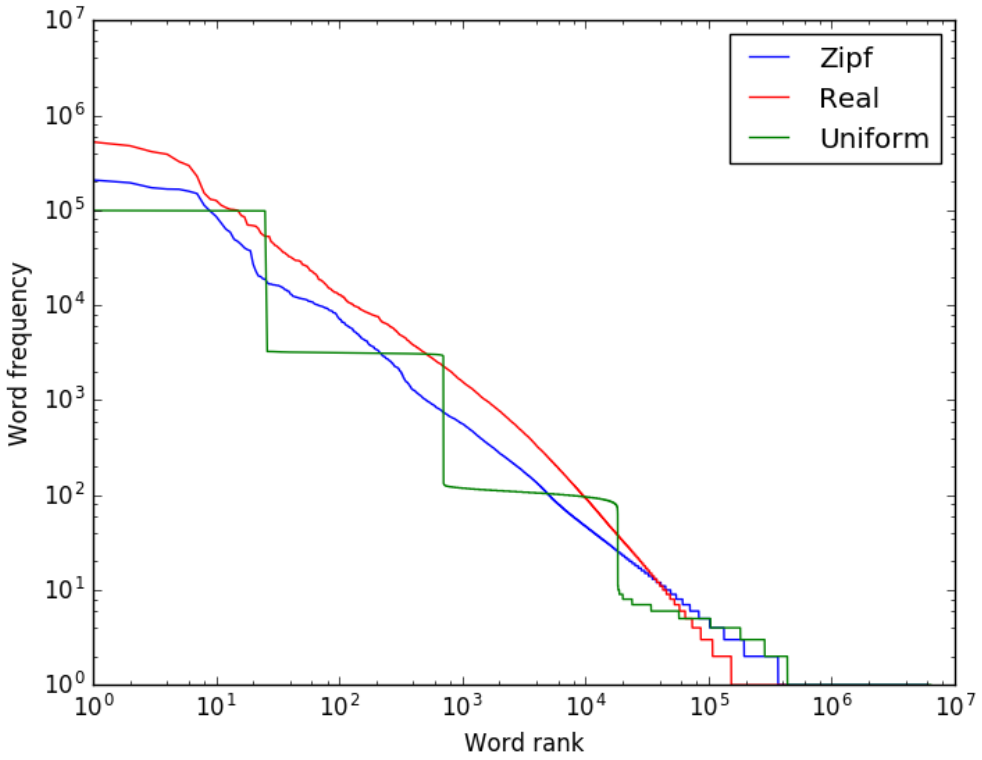
Let us assume you would want to creat a test corpus for some experiments. That test corpus has to have a similar word rank frequency diagram as the original data set. Which of the two generators would you use? You should perform the Kolmogorov Smirnov test as discussed in the lecture by calculating the 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?**

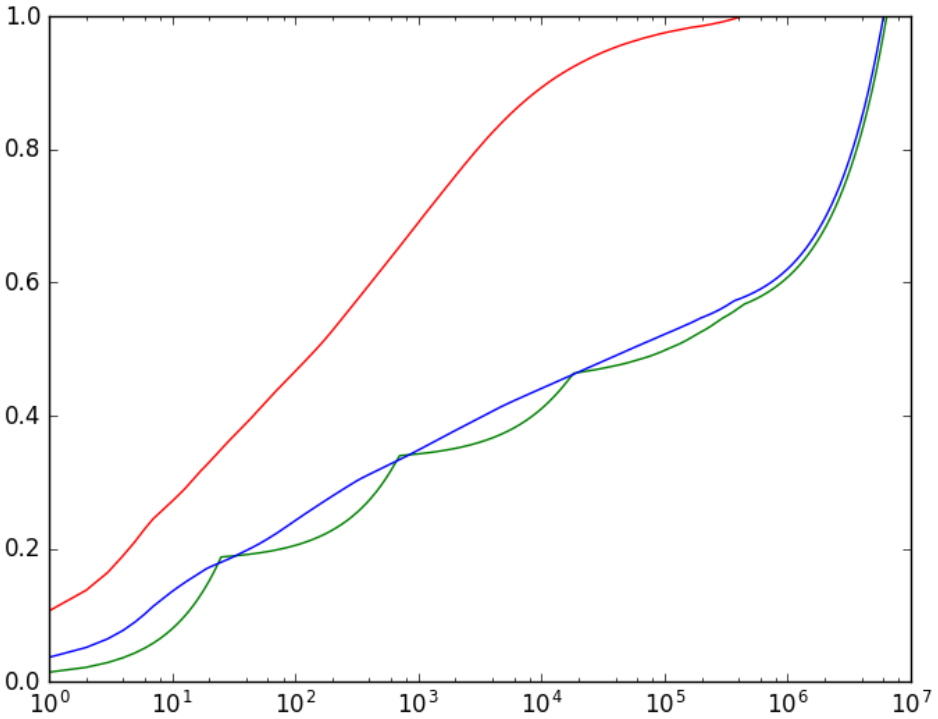
## 2.4 Hints:

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:**
**Modeling choice: We decided to ignore characters in SEW that are not letters or space.**
2.2
Word rank diagram:




CDF Diagram:




2.3
After initial run, these are the results:

1. Distance from real to zipf: 0.4644830643304992
2. Distance from real to uniform: 0.4831408811802278

Second run:

1. Distance from real to zipf: 0.4649174865652013
2. Distance from real to uniform: 0.4831021866265434

Consistently, generator based on the Zipf distribution proves to be better.

# 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

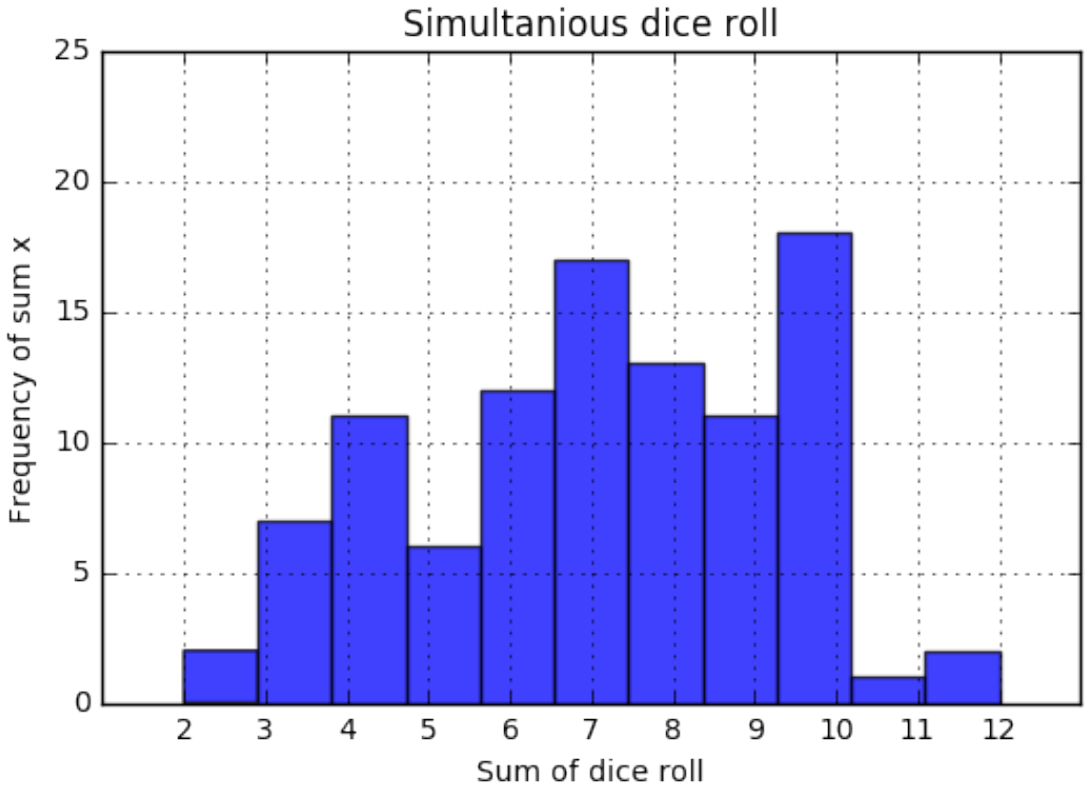
1. You can use function from the lecture to calculate rank and normalized cumulative sum for CDF.
2. Do not forget to give proper names of CDF plot axes or maybe even change the ticks values of x-axis.

## 3.2 Only for nerds and board students (0 Points)

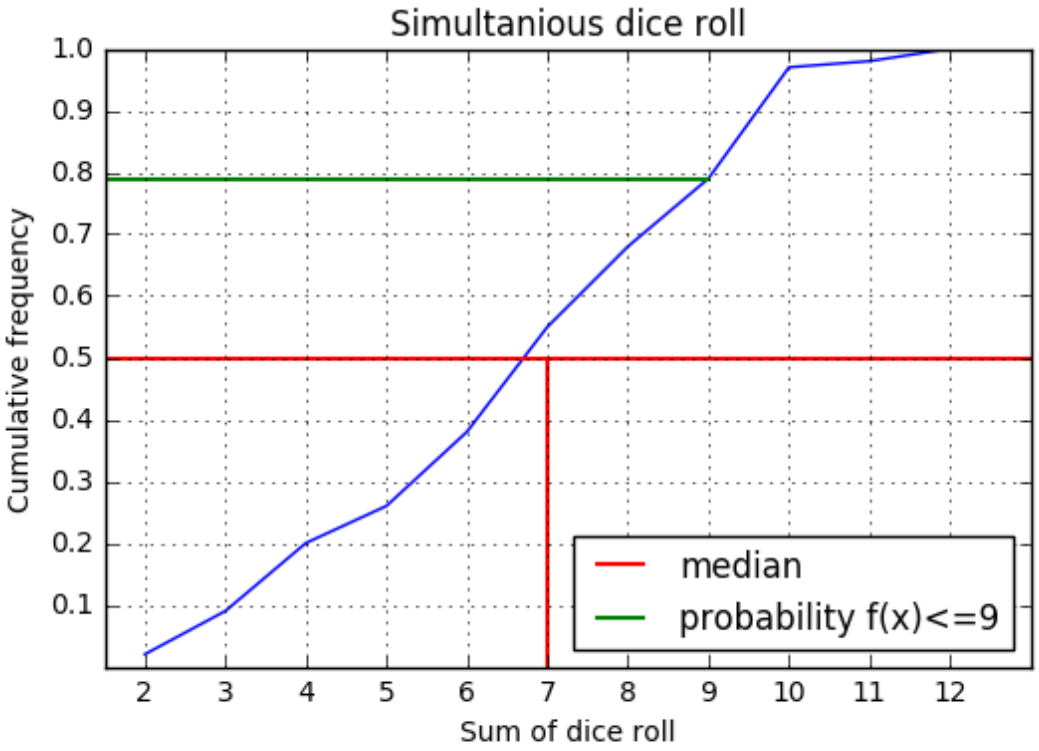
Assuming 20 groups of students. What is the likelihood that at least two groups come up with the same histograms in the case for n (=100)?

Answer:

1. Histogram for 100 times:


Simultanious dice roll
Frequency of sum x
Sum of dice roll
0
5
10
15
20
25
2
3
4
5
6
7
8
9
10
11
12

2. CDF:




```
Probability of f(x)<=9 is 0.79
```

Median is 7, and probability is 0.79

3.
```
In [15]: def perform_KS(dict1, dict2):
             maximum = 0
             for k,v in dict1.items():
                 maximum = abs(v - dict2[k]) if abs(v - dict2[k]) > maximum else maximum
             return maximum
```

```
In [16]: print(perform_KS(counter, counter2))

0.09999999999999998
```

4. With 1000 runs:

```
print(perform_KS(counter, counter2))
```

```
0.0619999999999997
```

5. Increasing the number of runs makes the distance smaller, as this distribution will be more similar to a normal distribution.

Code: task 1:

```
d1 = "this is a text about web science".split(" ")
d2 = "web science is covering the analysis of text corpora".split(" ")
d3 = "scientific methods are used to analyze webpages".split(" ")

# Get all the words in a corpus
sve = d1.copy()
sve.extend(d2)
sve.extend(d3)
sve = set(sve)


len(sve)


# Counting words in documents
def brojac(d1):
    d1_dic = {}
    for elem in sve:
#         print(elem)
        if elem in d1:
#             print(d1)
            d1_dic[elem] = 1
        else:
            d1_dic[elem] = 0
    return d1_dic
d1_dict = brojac(d1)
d2_dict = brojac(d2)
d3_dict = brojac(d3)

# Scalar product
def scalar(d1, d2):
    suma = 0
    for k,v in d1.items():
        suma += v * d2[k]
    return suma




from math import sqrt
def euclidian_len(d1):
    return sqrt(sum([v*v for k,v in d1.items()]))
def cosine(d1, d2):
    return (scalar(d1,d2)/(euclidian_len(d1)*euclidian_len(d2)))

print(cosine(d1_dict, d2_dict))
print(cosine(d1_dict, d3_dict))
print(cosine(d3_dict, d2_dict))
```

Code: task 2:

```
import pandas as pd
import re
from collections import Counter
from statistics import median, mean
data = []
with open('data.txt', 'r', encoding='utf-8') as f:
    for line in f:
```

```
        if line == "\n": continue
        data.append(line)
df = pd.DataFrame(data, columns=['article'])
# df['words'] = df['article'].apply(lambda x: re.findall(r'\w+', x))
# df['word_count'] = df['words'].apply(lambda x: len(x))
# df = df.drop(df[df.word_count == 0].index)

# Initial parsing of wikipedia data. All non letter characters are ignored.
df['article'] = df['article'].apply(lambda x: x.lower())
df['article'] = df['article'].apply(lambda x: "".join([p for p in x if p.isalpha() or p==" "]))
df["counts"] = df['article'].apply(lambda x: Counter(x))
df['char_count'] = df['counts'].apply(lambda x: sum([v for k,v in x.items()]))
# df['char_count'] = df['counts'].apply(lambda x: sum([v for k,v in x.items() if k.isalpha() or k==" "]))


real_data = " ".join(df["article"].values)


num_chars = sum(df['char_count'])

# Loading of probabilities
with open("probabilities.py.txt") as prob:
    dump = []
    for line in prob:
        if line != "\n":
            dump.append(line[:-1])


exec(dump[0])
exec(dump[1])
# zipf_probabilities, uniform_probabilities

# Creating cumulatives
kumul_zipf = {}
suma = 0
for k,v in zipf_probabilities.items():
    suma+= v
    kumul_zipf[suma] = k
kumul_unif = {}
suma = 0
for k,v in uniform_probabilities.items():
    suma+= v
    kumul_unif[suma] = k


def find_next(num, lista):
    for elem in lista:
        if elem >= num:
            return elem

import random

def generate_chars(number, CDF_dict):
    CDF = sorted(CDF_dict)
    return "".join(CDF_dict[find_next(random.random(), CDF)] for i in range(0,number))

# Generating corpuses
import time
f = open('unif_data.txt', 'w')
start = time.time()
unif_data = generate_chars(num_chars, kumul_unif)
end = time.time()
```

```
print("Time taken : {}".format(end - start))
f.write(unif_data)
f.close()
f = open('zipf_data.txt', 'w')
start = time.time()
zipf_data = generate_chars(num_chars, kumul_zipf)
end = time.time()
print("Time taken : {}".format(end - start))
f.write(unif_data)
f.close()


unif_words = re.findall(r'\w+', unif_data)
zipf_words = re.findall(r'\w+', zipf_data)

# Ranking words

unif_c = Counter(unif_words)
zipf_c = Counter(zipf_words)


words, frequencies_u = zip(*unif_c.most_common())



words, frequencies_z = zip(*zipf_c.most_common())



real_words = re.findall(r'\w+', real_data)
real_c = Counter(real_words)



words, frequencies_r = zip(*real_c.most_common())


# Saving generated frequencies
with open("generated.txt", 'w') as g:
    g.write("{}".format(frequencies_u))
    g.write('\n')
    g.write("{}".format(frequencies_z))
    g.write('\n')
    g.write("{}".format(frequencies_r))
    g.write('\n')



# Log log plotting
import matplotlib.pyplot as plt
plt.loglog([i for i in range(0,len(frequencies_z))], frequencies_z, label='Zipf')
plt.loglog([i for i in range(0,len(frequencies_r))], frequencies_r, color='red', label="Real")
plt.loglog([i for i in range(0,len(frequencies_u))], frequencies_u, color='green', label="Uniform")
plt.xlabel('Word rank')
plt.ylabel('Word frequency')
plt.legend(loc='upper right')
plt.show()

# Plotting cumulatives
import numpy as np
freq_r_sum = sum(frequencies_r)
frequencies_cum_r = list(map(lambda x: x/freq_r_sum, frequencies_r))
```

```
frequencies_cum_r = np.cumsum(frequencies_cum_r)



freq_z_sum = sum(frequencies_z)
frequencies_cum_z = list(map(lambda x: x/freq_z_sum, frequencies_z))
frequencies_cum_z = np.cumsum(frequencies_cum_z)
freq_u_sum = sum(frequencies_u)
frequencies_cum_u = list(map(lambda x: x/freq_u_sum, frequencies_u))
frequencies_cum_u = np.cumsum(frequencies_cum_u)


# Calculating max distance in CDF
dist_r_z = np.absolute(np.subtract(frequencies_cum_r, frequencies_cum_z[:len(frequencies_cum_r)]))
max_dist_r_z = max(dist_r_z)
max_index_r_z = np.where(dist_r_z == max_dist_r_z)
max_index_r_z = max_index_r_z[0][0]



dist_r_u = np.absolute(np.subtract(frequencies_cum_r, frequencies_cum_u[:len(frequencies_cum_r)]))
max_dist_r_u = max(dist_r_u)
max_index_r_u = np.where(dist_r_u == max_dist_r_u)
max_index_r_u = max_index_r_u[0][0]


plt.semilogx([i for i in range(0,len(frequencies_cum_r))], frequencies_cum_r, 'r')
plt.semilogx([i for i in range(0,len(frequencies_cum_u))], frequencies_cum_u, 'g')
plt.semilogx([i for i in range(0,len(frequencies_cum_z))], frequencies_cum_z, 'b')
# plt.plot([max_index_r_z,max_index_r_z], [0,1], 'b-')
axes = plt.gca()
# axes.set_xlim([xmin,xmax])
axes.set_ylim([0,1])
plt.show()
```

Code: task 3:

```
import random
import matplotlib.pyplot as plt
def roll_die():
    return random.randint(1,6)


def roll_2_dies(n_times):
    arr = []
    for i in range(0,n_times):
        x = roll_die()
        y = roll_die()
        arr.append(x+y)
    return arr


# times_to_run = 100
times_to_run = 1000
arr = roll_2_dies(times_to_run)
arr2 = roll_2_dies(times_to_run)


n, bins, patches = plt.hist(arr, bins=11, normed=0, facecolor='blue', alpha=0.75)
plt.xlabel('Sum of dice roll')
plt.ylabel('Frequency of sum x')
```

```
plt.title('Simultanious dice roll')
plt.xticks([i for i in range(2,13)])
plt.axis([1, 13, 0, 250])
plt.grid(True)

plt.show()



from collections import Counter
counter = Counter(arr)
counter2 = Counter(arr2)
print(counter)
print(counter2)


for k,v in counter.items():
    counter[k] = v/times_to_run
for k,v in counter2.items():
    counter2[k] = v/times_to_run


suma = 0
for k,v in counter.items():
    suma+= v
    counter[k] = suma
suma = 0
for k,v in counter2.items():
    suma+= v
    counter2[k] = suma


x_tics = []
y_tics = []
for k,v in counter.items():
    x_tics.append(k)
    y_tics.append(v)


from statistics import median
med = median(arr)
bar = plt.plot(x_tics, y_tics)
plt.xlabel('Sum of dice roll')
plt.ylabel('Cumulative frequency')
plt.title('Simultanious dice roll')
plt.xticks(x_tics)
plt.yticks([0.1*i for i in range(1,11)])
plt.axis([1.5, 13, 0, 1.0])
plt.grid(True)


plt.plot([0, 13], [0.5, 0.5], 'r-', linewidth=1.5)
plt.plot([med, med], [0, 0.5], 'r-', linewidth=1.5, label="median")
plt.plot([0, 9], [y_tics[7], y_tics[7]], 'g-', linewidth=1.5, label="probability f(x)<=9")
plt.legend(loc="lower right")

plt.show()
print("Probability of f(x)<=9 is {}".format(y_tics[7]))


print(med)

```

```
87
88  def perform_KS(dict1, dict2):
89      maximum = 0
90      for k,v in dict1.items():
91          maximum = abs(v - dict2[k]) if abs(v - dict2[k]) > maximum else maximum
92      return maximum
93
94  print(perform_KS(counter, counter2))
```

# 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`.