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: India
Team members: Jasvinder Kaur, Amani Gaddamedi, Jalpa Patel.

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:
Documents 1 and Document 2 should be the most similar because:

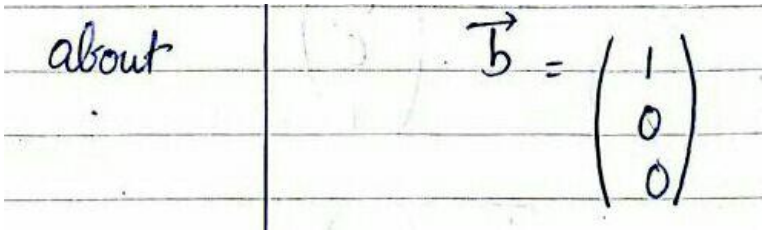
1. no documents other than these two have any words in common.E.g just looking at the document, we can just notice that they have 4 words in common.

2. Also, documents D1 and D2 are similar meaningfully also as D1 introduces web science and D2 further explains web science explictly, while in D3 there is nothing mentioned of web science explicitly.

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:
   Here, 19 different words are present in 3 documents of the given corpus. So, 19 base vectors would be needed, each representing a word.
   E.g The word "a represents base vector a, "about" represents base vector b and so on, as shown in figure below:



2. What does each dimension of the vector space stand for?
   Answer:
   Each dimension of the vector space stands for the respective base vector which further represents respective word.

3. How many dimensions does the vector space have?
   Answer:
   The base vector correspond to the dimension or direction of the vector space. So, 19 dimensions are present in the vector space, as there are 19 base vectors present

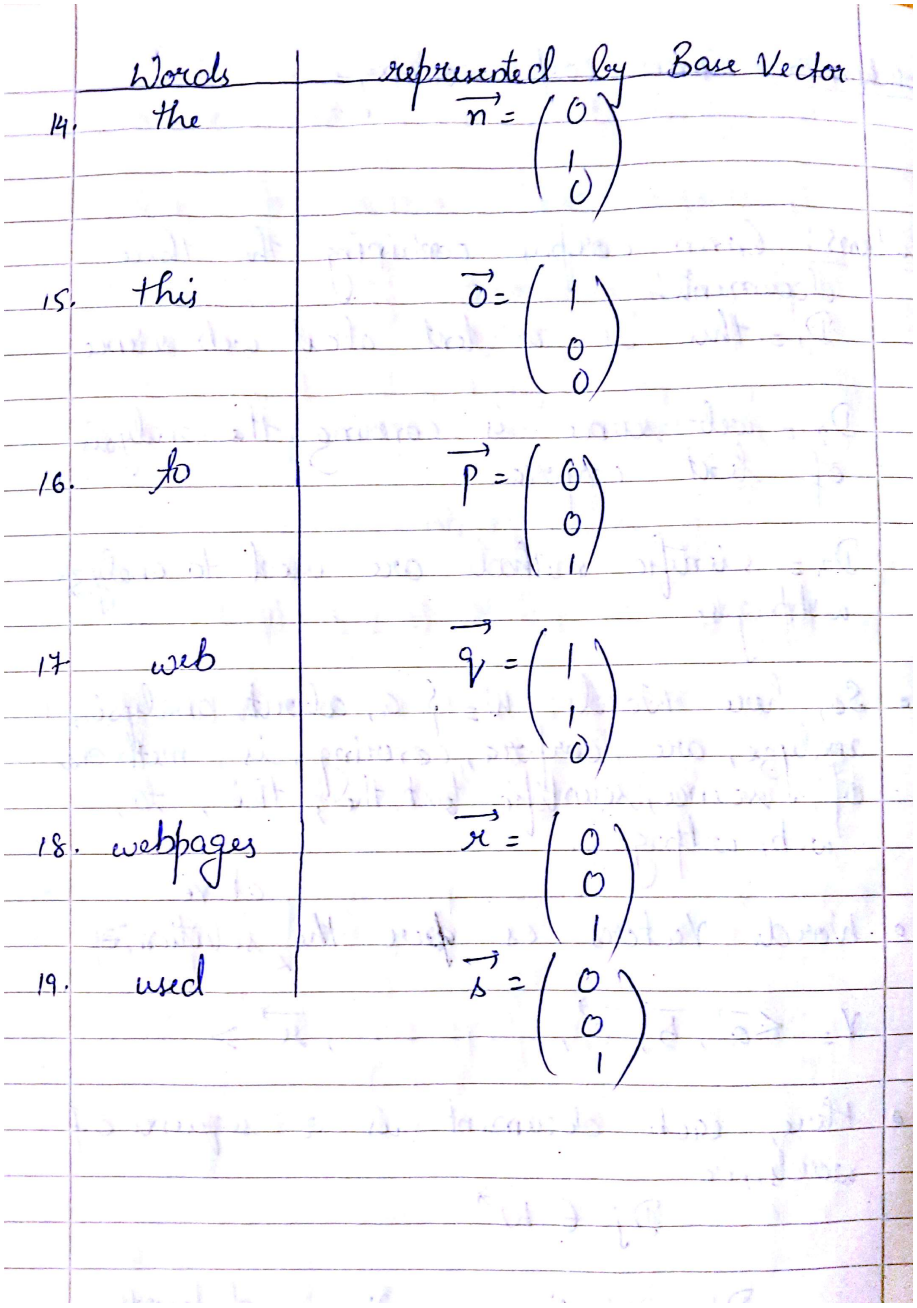
4. Create a table to map words of the documents to the base vectors.
   Answer:

Answer 2: ~~If~~ Each dimension

Answer 4:

	Words	represented by Base Vector
1.	a	$\vec{a} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$
2.	about	$\vec{b} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$
3.	analysis	$\vec{c} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$
4.	analyze	$\vec{d} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
5.	are	$\vec{e} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
6.	corpora	$\vec{f} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$

	Words	represented by Base vector
7.	covering	$\vec{g} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$
8.	is	$\vec{h} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$
9.	methods	$\vec{i} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
10.	of	$\vec{j} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$
11.	science	$\vec{k} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$
12.	scientific	$\vec{l} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
13.	text	$\vec{m} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$



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:

Solution 5: Given corpus ~~containg~~

Solutions: Given corpus containing the 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.

- So, here Words $W = \{$ a, about, analysis, analyze, are, corpora, covering, is, methods, of, science, scientific, text, the, this, to, web, webpages$\}$.
- Words Vectors as per the above solution: ~~of~~

$V = \langle \vec{a}, \vec{b}, \vec{c}, \ldots\ldots, \vec{x} \rangle$

- Here, each document is a sequence of words, i.e.

$$D_j \in W^*$$

$D_j = w_{i_1} w_{i_2} \ldots w_{i_m}$ of length m.

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

where tf denotes term frequency.

$tf(w_i, D_j)$ => term frequency of a word w_i in Document D_j.

Document frequency is defined as

$df(w_i) = |\{D_j \mid w_i \text{ in } D_j\}|$

Term frequency table:

	Word w_i	Document D_j		
		D_1	D_2	D_3
1.	a	1	0	0
2.	about	1	0	0
3.	analysis	0	1	0
4.	analyze	0	0	1
5.	are	0	0	1
6.	corpora	0	1	0
7.	covering	0	1	0
8.	is	1	1	0
9.	methods	0	0	1
10.	of	0	1	0
11.	science	1	1	0
12.	scientific	0	0	1
13.	text	1	1	0
14.	the	0	1	0

Word w_i	Document D_j		
	D_1	D_2	D_3
this	1	0	0
to	0	0	1
web	1	1	0
webpages	0	0	1
used	0	0	1

Let us now create the document vectors.

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

$$= tf(\text{this}, D_1)\vec{o} + tf(\text{is}, D_1)\vec{h} + tf(a, D_1)\vec{a} + tf(\text{text}, D_1)\vec{m} + tf(\text{about}, D_1)\vec{b} + tf(\text{web}, D_1)\vec{q} + tf(\text{science}, D_1)\vec{k}$$

$$= 1\begin{pmatrix}1\\0\\0\end{pmatrix} + 1\begin{pmatrix}1\\1\\0\end{pmatrix} + 1\begin{pmatrix}1\\0\\0\end{pmatrix} + 1\begin{pmatrix}1\\1\\0\end{pmatrix} + 1\begin{pmatrix}1\\0\\0\end{pmatrix} + 1\begin{pmatrix}1\\1\\0\end{pmatrix} + 1\begin{pmatrix}1\\1\\0\end{pmatrix}$$

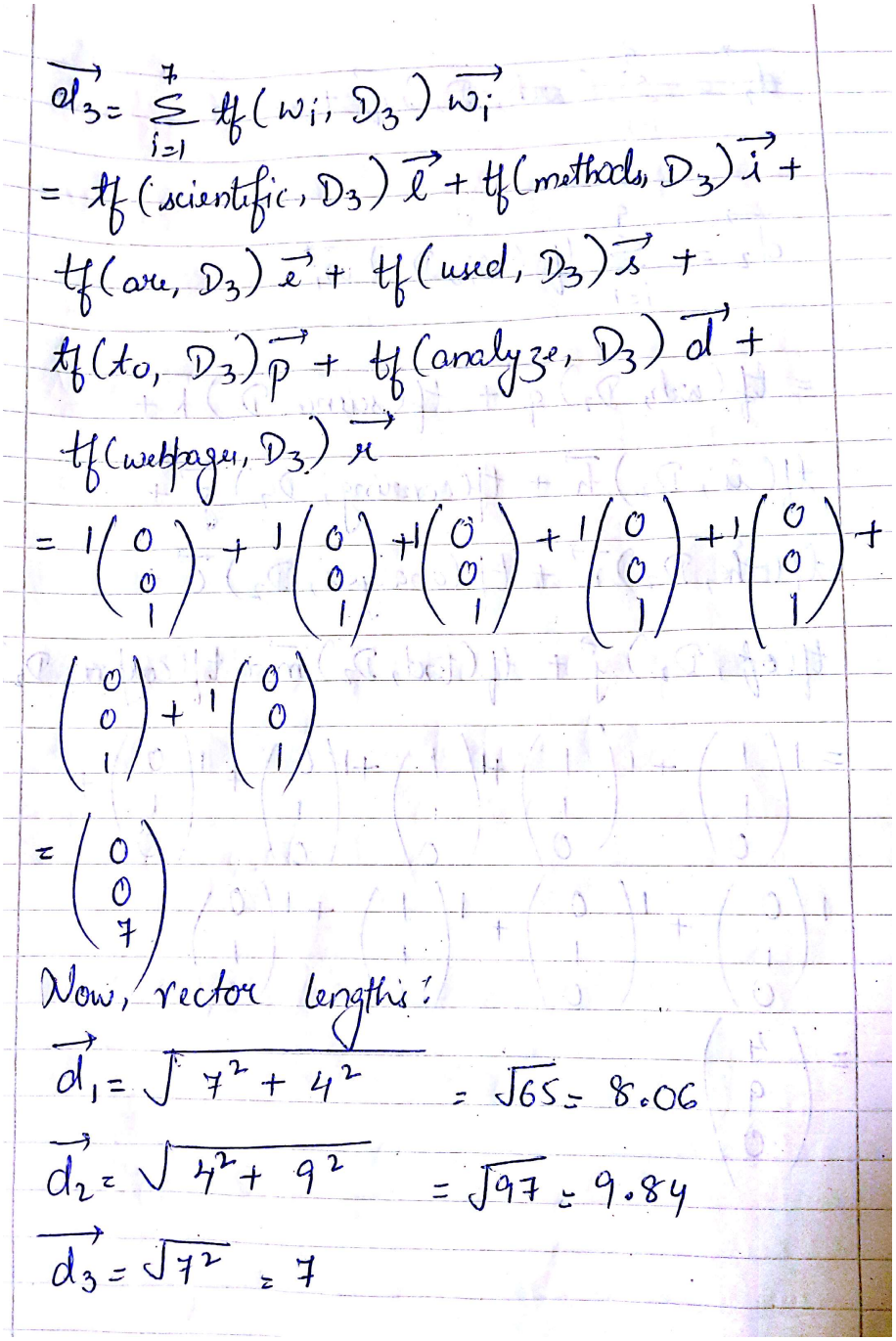
$$= \begin{pmatrix}7\\4\\0\end{pmatrix}$$

$$\vec{d_2} = \sum_{i=1}^{9} tf(w_i, D_2)\,\vec{w_i}$$

$$= tf(web, D_2)\vec{q} + tf(science, D_2)\vec{k} + tf(is, D_2)\vec{h} + tf(covering, D_2)\vec{g} + tf(the, D_2)\vec{n} + tf(analysis, D_2)\vec{c} + tf(of, D_2)\vec{j} + tf(text, D_2)\vec{m} + tf(corpora, D_2)\vec{f}$$

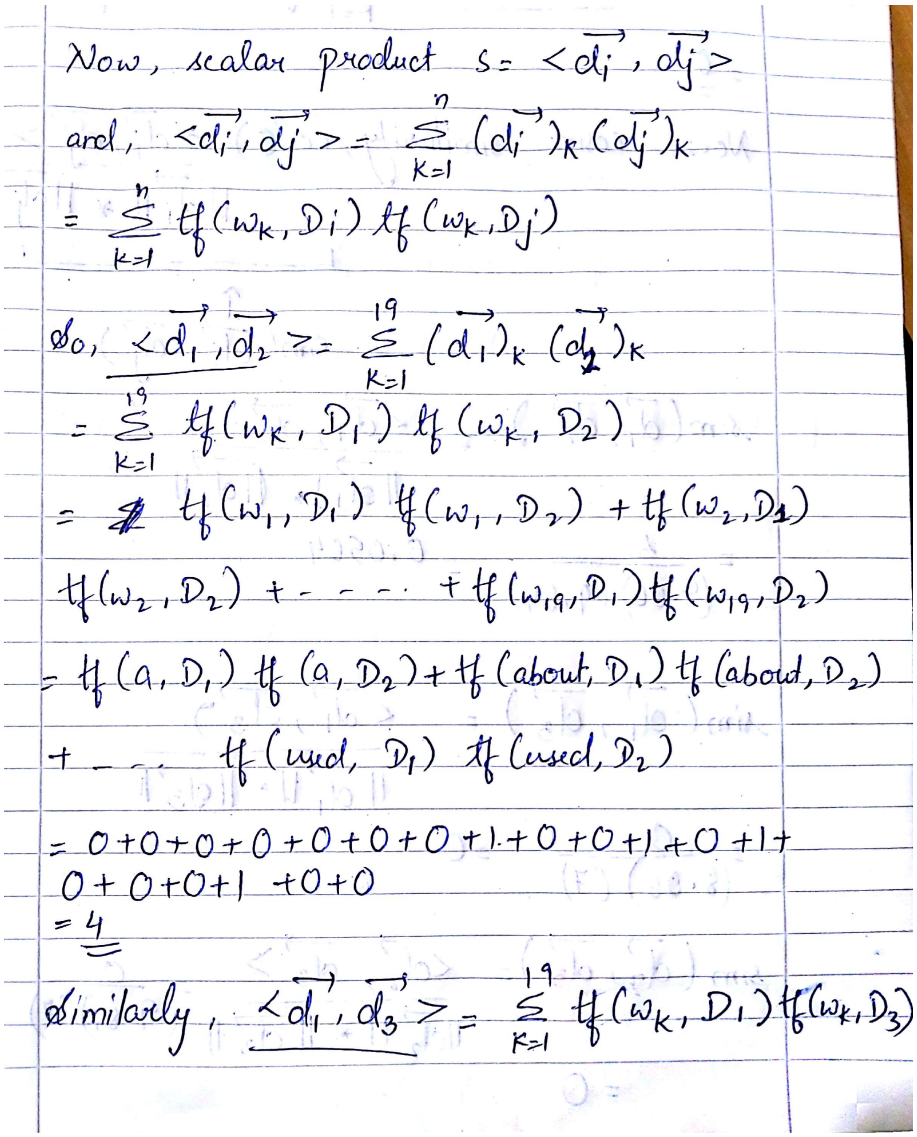
$$= 1\begin{pmatrix}1\\1\\0\end{pmatrix} + 1\begin{pmatrix}1\\1\\0\end{pmatrix} + 1\begin{pmatrix}1\\1\\0\end{pmatrix} + 1\begin{pmatrix}0\\1\\0\end{pmatrix} + 1\begin{pmatrix}0\\1\\0\end{pmatrix} + 1\begin{pmatrix}0\\1\\0\end{pmatrix} + 1\begin{pmatrix}0\\1\\0\end{pmatrix} + 1\begin{pmatrix}1\\1\\0\end{pmatrix} + 1\begin{pmatrix}0\\1\\0\end{pmatrix}$$

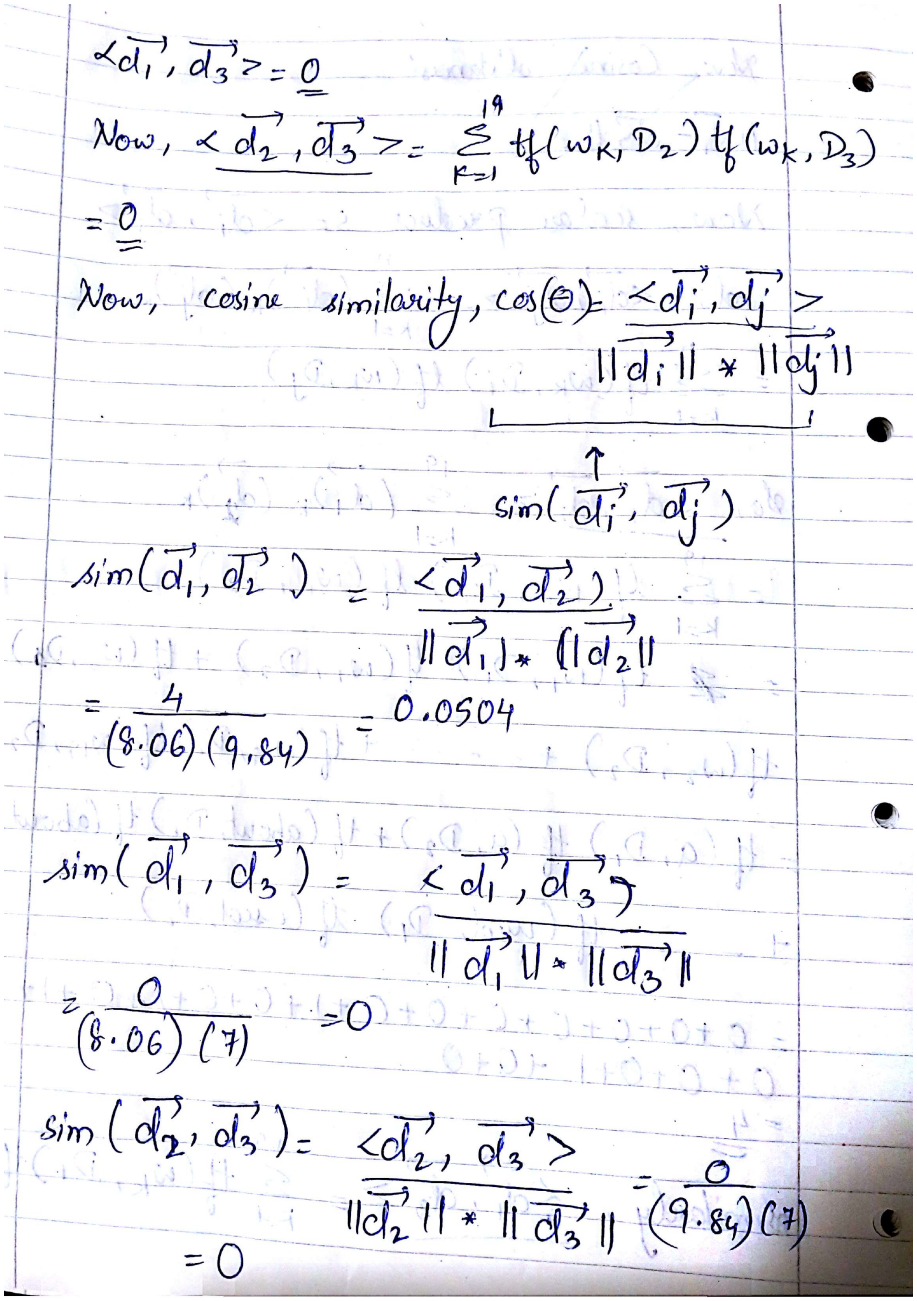
$$= \begin{pmatrix}4\\9\\0\end{pmatrix}$$



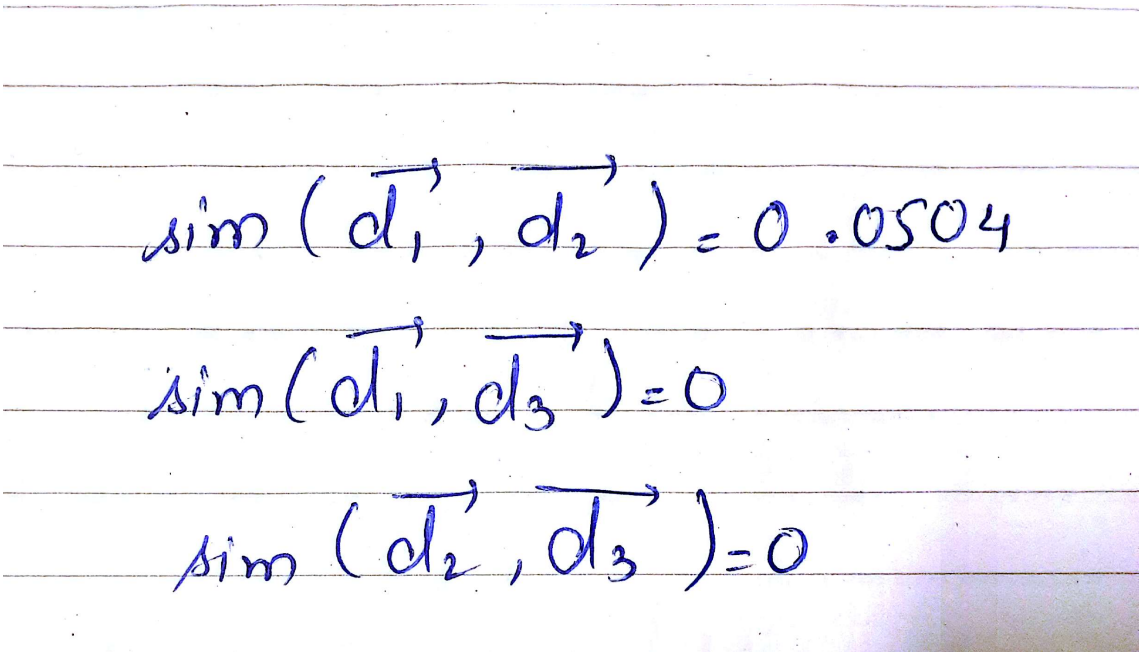
6. Calculate the cosine similarity between all three pairs of vectors.
   Answer:

Cosine similarity is calculated as follows:





7. According to the cosine similarity which 2 documents are most similar according to the constructed model.
   Answer:
   According to the similarity values, the final order in which the documents are presented as result to the query are:



So, document 1 and document 2 are the most similar documents.

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:
Yes, the results of the model match our expectations from the first subtask.

1. Each document in the collection is represented as a vector in a vector space.

2. The documents that are close to each other in this space are semantically(referring to semantics) similar i.e. less is the deviation between two vectors, more similar they are and vice-versa.

3. This closeness is calculated by the cosine of the angle between them.

4. Documents are ranked by decreasing cosine value.

5. sim(d1,d2) = 1 when d1 = d2 and sim(d1,d2) = 0 when d1 and d2 have no common terms.

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:

Running our python code:

India_assignment7_2.py is the main file to be run.
India_process_data.py is imported and used in the main file.
probabilities.py contains the provided two probability distributions.

a) probabilities.py

```
1: zipf_probabilities = {' ': 0.17840450037213465, '1': 0.004478392057619917
2:
3: uniform_probabilities = {' ': 0.1875, 'a': 0.03125, 'c': 0.03125, 'b': 0.
```

b) India_process_data.py

```
1: def get_freq(word_list):
2:   frequency_of_word_size = {}
3:   for word in word_list:
4:     frequency_of_word_size[word] = frequency_of_word_size.get(word, 0) +
5:   return frequency_of_word_size
6:
7: def read_and_countwords(text):
8:   words_count = {}
9:   text.strip()
10:  cleandot_data = text.replace('.',' ')
11:  cleancomma_data = cleandot_data.replace(',',' ')
12:  cleanbracket1_data = cleancomma_data.replace('(',' ')
13:  cleanbracket2_data = cleanbracket1_data.replace(')',' ')
14:  cleancolon_data = cleanbracket2_data.replace(':',' ')
15:  cleanspace_data = cleancolon_data.replace(' ',' ')
16:  cleanquotes_data = cleanspace_data.replace('"',' ')
17:  word_list = cleanquotes_data.split()
18:  words_count = get_freq(word_list)
19:  return words_count
20:
21: def word_frequency_in_file(file_name):
22:   with open(file_name,encoding="utf-8") as file:
23:     return read_and_countwords(file.read())
24:
25:
26: def count_chars_and_spaces(file_name):
27:   count_chars_and_spaces = 0
28:   with open(file_name, encoding="utf-8") as file:
29:     data = file.read()
30:   for char in data:
31:     if char.isalpha() or char.isspace():
32:       count_chars_and_spaces += 1
33:   return count_chars_and_spaces
```

```
34:
35:
36: if __name__ == "__main__":
37: a =count_chars_and_spaces("simple-20160801-1-article-per-line")
```

c) India_assignment7_2.py

```
1: #@author: Jasvinder
2: import probabilities as prob
3: import India_process_data as data
4: import numpy as np
5: import matplotlib.pyplot as plt
6: import os.path
7: # samples n characters from the distribution
8: #input:
9: #   probalities: Dictionary - required data to be sampled
10: #   n: Integer: - No of characters to be sampled
11: #output:
12: #   String - generated value after sampling
13: def sample_n_times(probabilities, n):
14:   chars = list(probabilities.keys())
15:   prob =  [probabilities[char] for char in chars]
16:   return "".join(np.random.choice(chars, n, p = prob))
17:
18: # finds maximum pointwise distance of the Cdfs
19: # input:
20: #   Cdfs of the two generated texts
21: # output:
22: #    Integer - maximum pointwise distance
23: def find_norm(y1, y2):
24:   small_list_size = min(y1.shape[0], y2.shape[0])
25:   return np.max(np.abs(np.subtract
26:   (y1[:small_list_size], y2[:small_list_size])))
27:
28: # finds the word rank frequency
29: # input:
30: #   word frequency in file
31: # output:
32: #   rank of the word frequency
33: def rank_freq(freq):
34:     points = [(k,v) for k, v in freq.items()]
35:     dtype = [('x', 'unicode'), ('y', int)]
36:     np_points = np.array(points, dtype = dtype)
37:     np_points['y'] *= -1
38:     sorted_freq = np.sort(np_points, order='y')
39:     sorted_freq['y'] *= -1
40:     return sorted_freq
41:
42: if __name__ == '__main__':
43:   generated_file_1 = "generated_1.txt"
```

```
44:  generated_file_2 = "generated_2.txt"
45:  wikipedia_file = "simple-20160801-1-article-per-line"
46:  if not os.path.isfile(generated_file_1) or not os.path.isfile(generated
47:    total_chars = data.count_chars_and_spaces(wikipedia_file)
48:
49:    generated_string = sample_n_times(prob.uniform_probabilities, total_c
50:
51:    if not os.path.isfile(generated_file_1):
52:      with open(generated_file_1, "w") as file:
53:        file.write(generated_string)
54:    generated_string = sample_n_times(prob.zipf_probabilities, total_char
55:    if not os.path.isfile(generated_file_2):
56:      with open(generated_file_2, "w") as file:
57:        file.write(generated_string)
58:  plt.title(
59:  "plot of Rank vs frequency  of wiki and two generated texts.")
60:  plt.xlabel('Rank')
61:  plt.ylabel('freq')
62:
63:  fre = data.word_frequency_in_file(wikipedia_file)
64:  y_wiki = rank_freq(fre)['y']
65:  x_wiki = np.arange(1, len(y_wiki) + 1, 1)
66:  red_line, = plt.loglog(x_wiki, y_wiki,"r-", label="simple english wikip
67:
68:  fre = data.word_frequency_in_file(generated_file_1)
69:  y_g1 = rank_freq(fre)['y']
70:  x_g1 = np.arange(1, len(y_g1) + 1, 1)
71:  blue_line, = plt.loglog(x_g1, y_g1,"b-", label="uniform probability gen
72:
73:  fre = data.word_frequency_in_file(generated_file_2)
74:  y_g2 = rank_freq(fre)['y']
75:  x_g2 = np.arange(1, len(y_g2) + 1, 1)
76:  green_line, = plt.loglog(x_g2, y_g2, "g-", label="zipf probability gene
77:  plt.legend(handles=[red_line, blue_line, green_line],
78:  fontsize = 8, framealpha = 0.85, frameon = True )
79:  plt.show()
80:
81:  plt.title(
82:  "cumulated plots of Rank frequency plot of wiki and two generated texts
83:  plt.xlabel('Rank')
84:  plt.ylabel('p(Frequency >= n)')
85:
86:  csum_y_wiki = np.cumsum(y_wiki)
87:  cdf_y_wiki = csum_y_wiki/np.max(csum_y_wiki)
88:
89:  red_line, = plt.plot(x_wiki, cdf_y_wiki, "r-",
90:  label="simple english wikipedia")
91:
92:  csum_y_g1 = np.cumsum(y_g1)
```

```
93:   cdf_y_g1 = csum_y_g1 / np.max(csum_y_g1)
94:
95:   blue_line, = plt.plot(x_g1, cdf_y_g1, "b-", label="uniform probab")
96:   plt.xscale('log')
97:
98:   csum_y_g2 = np.cumsum(y_g2)
99:   cdf_y_g2 = csum_y_g2 / np.max(csum_y_g2)
100:
101:  green_line, = plt.plot(x_g2, cdf_y_g2, "g-", label="zipf probab")
102:  plt.xscale('log')
103:
104:  plt.legend( handles = [red_line, blue_line, green_line],
105:  fontsize = 8, framealpha = 0.85, frameon = True )
106:  plt.show()
107:
108:  smirnov_uniform = find_norm(cdf_y_wiki, cdf_y_g1)
109:  smirnov_zipf = find_norm(cdf_y_wiki, cdf_y_g2)
110:  print("Kolmogorov Smirnov test for g1")
111:  print(smirnov_uniform)
112:  print("Kolmogorov Smirnov test for g2")
113: print( smirnov_zipf)
```

d) Kolmogorov Smirnov Test:
When we generate the two text corpora for a second time, we have different values of Kolmogorov Smirnov test each time because of the random process sampling, So, each time we have different values because of this randomization.
Here are the results:
Kolmogorov Smirnov test for g1 0.478948511954
Kolmogorov Smirnov test for g2 0.457976843257
Here g1 represents generated text from one generated text and g2 is of another generated text and g1 is more closer to the original data.
Output:

Figure 1: Word frequency diagram for generated text1

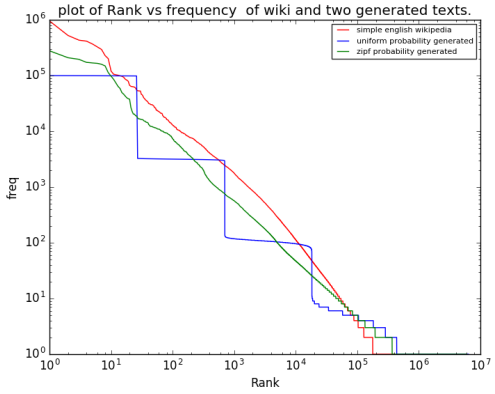


Figure 2: Word frequency diagram for generated text2

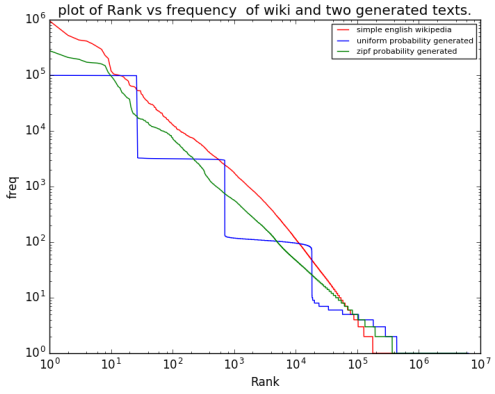


Figure 3: CDF Plot1

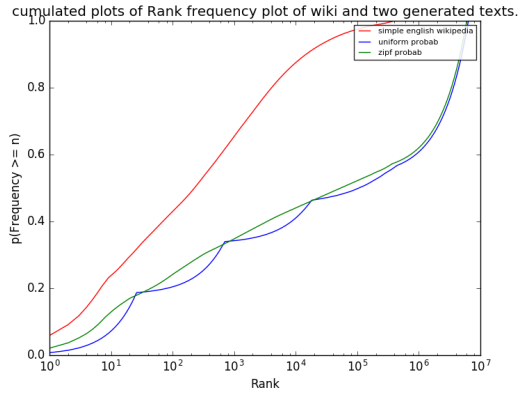
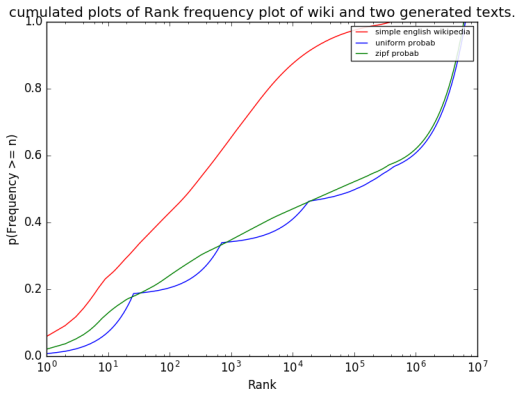


Figure 4: CDF Plot2



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.
   Answer: refer output images

   What is the probability of dice sum to be equal or less than 9? Mark the point on the plot.
   Answer:refer output images

4. Repeat the simulation a second time and compute the maximum point-wise distance of both CDFs.
   Answer:
5. Now repeat the simulation (2 times) with n=1000 and compute the maximum point-wise distance of both CDFs. Answer: refer output images

6. What conclusion can you draw from increasing the number of steps in the simulation?
   Answer:
   Increasing the number of steps in the simulation still results more or less in the equal values of medians and Cdfs. Sampling error will vary while running such a random process several times. The values for probabilities equal or less than 9 and that of maximum poitnwise distance are different each time.

3.1 Hints

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

Answer:

Running our python code:
India_assignment7_3.py is the file to be run.

1. probabilities.py

```
1: import numpy as np
2: import matplotlib.pyplot as plt
3: # rolls 2 dices simultaneously
4: # input:
5: #   no_of_times: Integer - no of times to roll the dice
6: # output:
7: #   Integer - sum of both dices' outputs
8: def roll_2_die(no_of_times):
9:   die_1 = np.random.random_integers(1, 6, no_of_times)
10:  die_2 = np.random.random_integers(1, 6, no_of_times)
11:  return die_1 + die_2
12:
13: # finds maximum pointwise distance
14: def find_norm(y1, y2):
15:  small_list_size = min(y1.shape[0], y2.shape[0])
16:  return np.max(np.abs(np.subtract
17:  (y1[:small_list_size], y2[:small_list_size])))
18:
19: # finds the median sum of two dice sides
20: def get_median(x, y):
21:  return x[np.min(np.where(y >= 0.5))]
22:
23: # finds the probability of dice sum to be <= to the required number
24: #given by prob passed here
25: def get_prob(x,y, prob):
26:  x_index = np.max(np.where(x <= prob))
27:  return y[x_index]#[x[np.max(np.where(x < 0.5))],np.max(y[np.where(x <=0.5)]
28: if __name__ == "__main__":
29:  dice_sum_output = roll_2_die(100)
30:  plt.title(
31:  "cumulated plots of Rank frequency plot of throwing two dice")
32:  plt.xlabel('dice output')
33:  plt.ylabel('Frequency')
34:  y = plt.hist(dice_sum_output, bins=11,normed= True, cumulative= True,
35:  histtype= "step", label="n=100" )
36:  a = get_median(y[1], y[0])
37:  print("median for n =100")
38:  print(a)
39:  x_9 = get_prob(y[1],y[0], 9)
40:  print("value of p <= 9 for n = 100")
41:  print(x_9)
42:  y1 = y[0]
43:  plt.plot([a , a],[0, .5],"m--",label = "median n=100")
```

```
44:   plt.plot([9 , 9],[0,x_9],"k--",label = "p < 9 n=100")
45:   dice_sum_output = roll_2_die(1000)
46:   y = plt.hist(dice_sum_output, bins=11,normed= True, cumulative= True,
47:   histtype= "step", label="n=1000" )
48:   print("median for n =1000")
49:   a = get_median(y[1], y[0])
50:   print(a)
51:   x_9 = get_prob(y[1],y[0], 9)
52:   print("value of p <= 9 for n = 1000")
53:   print(x_9)
54:   y2 = y[0]
55:   plt.plot([a , a],[0, .5],"y-",label = "median n=1000")
56:   plt.plot([9 , 9],[0,x_9],"r-",label = "p < 9 n=1000")
57:   plt.legend(fontsize = 8, framealpha = 0.85 )
58:   print("maximum pointwise distance")
59:   smirnov = find_norm(y1,y2)
60:   print(smirnov)
61: plt.show()
```

2. Output:

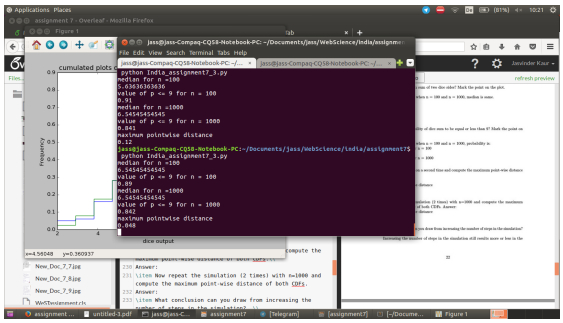
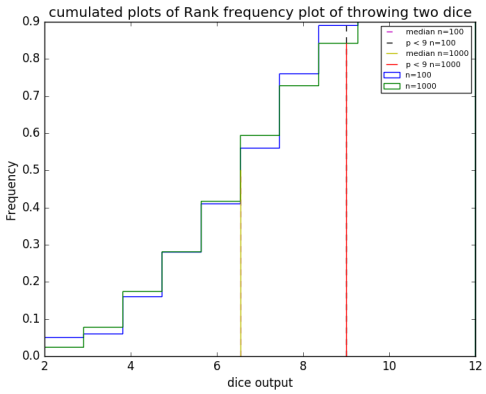


Figure 5: Word frequency diagram for generated text1



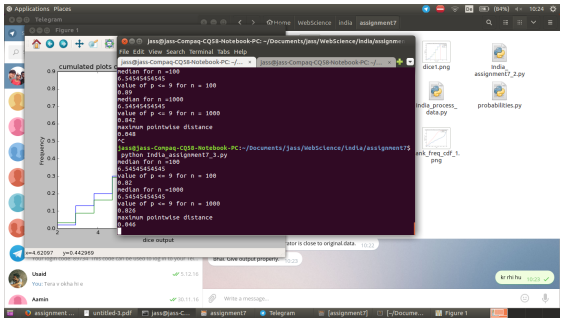
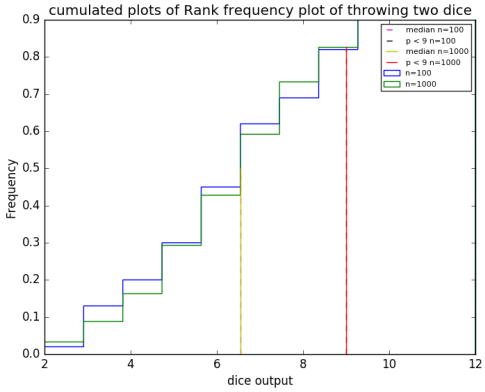


Figure 6: Word frequency diagram for generated text1



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`.