Introduction to Web Science

Assignment 6

Jun Sun
junsun@uni-koblenz.de

Iryna Dubrovska
idubrovska@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science
University of Koblenz-Landau

Submission until: December 14, 2021, 24:00 CET
Tutorial on: December 16, 2021, 16:00 CET

This assignment focuses on the concepts of 1) Advanced Statistic Models, 2) Modelling Similarity and 3) Programming in Python. Some of the tasks may require you to do additional research extending the lecture. Please keep the citation rules in mind. For all the assignment questions that require you to write a 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.

Date: 14/12/2021

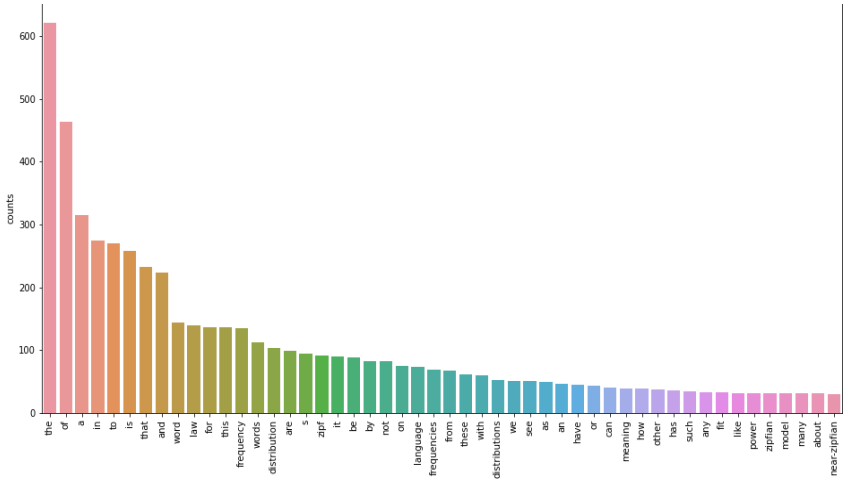
Team Name: Boehm

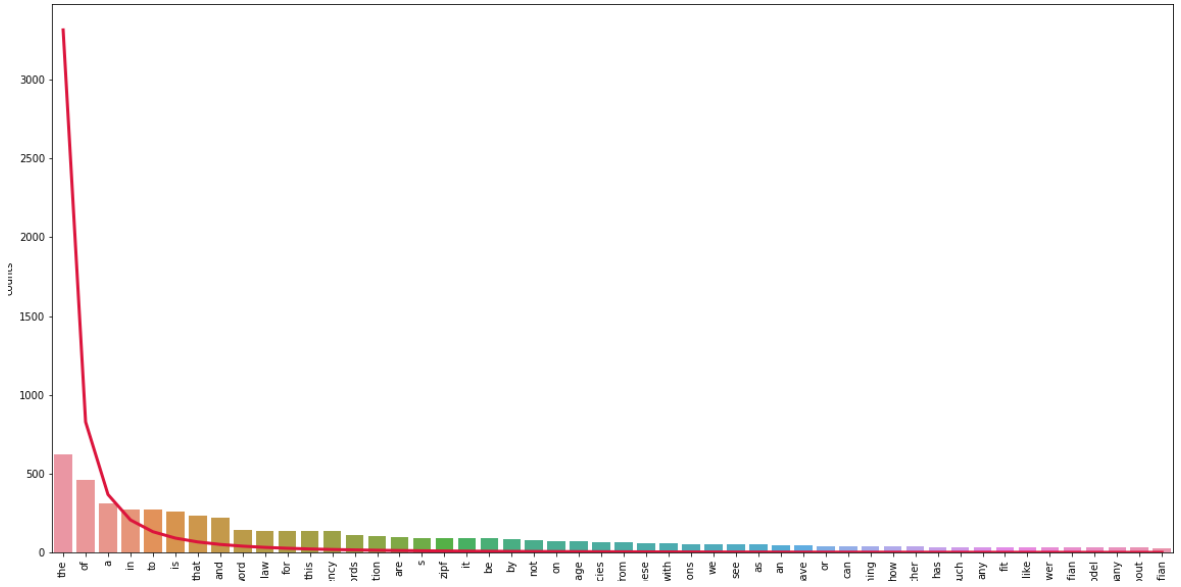
Abhinav Ralhan (abhinavr8@uni-koblenz.de)
Fatima Akram (fatimaakram9396@uni-koblenz.de)
Hammad Ahmed (hammadahmed@uni-koblenz.de)
Vishal Vidhani (vvidhani@uni-koblenz.de)

1 Python Programming. Zipf ’s Law (15 points)T Requests (18 points)

```
import re
import nltk
import random
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pylab import  rcParams
from scipy.stats import zipf
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize

# Fetching all data

all_data = ''
for files in ['Text_0123/Text_0.txt', 'Text_0123/Text_1.txt',
'Text_0123/Text_2.txt', 'Text_0123/Text_3.txt']:
    f = open(files, 'r')
    data = f.read()
    all_data += data

# Creates sentence tokens and fetches the cleaned data

sentence_tokens = sent_tokenize(all_data)
for idx, sentence in enumerate(sentence_tokens):
    sentence_tokens[idx] = sentence_tokens[idx].replace('\n', ' ')

while True:
    sentence_tokens = [sentence for sentence in sentence_tokens if
sentence != '']
    for idx, sentence in enumerate(sentence_tokens):
        if "Eq."in sentence or "eg." in sentence or "etc." in sentence:
            sentence_tokens[idx] = ' '.join(sentence_tokens[idx:idx+2])
            try:
                sentence_tokens[idx+1] = ''
            except:
                pass
    if '' not in sentence_tokens:
        break
all_data = ' '.join(sentence_tokens)

# Creates word tokens from cleaned sentences and words

word_tokens = word_tokenize(all_data)

cleaned_word_tokens = []

for word in word_tokens:
    if re.sub('[^A-Za-z]+', '', word) != '':
        cleaned_word_tokens.append(word)
```

```
cleaned_words = [word.lower() for word in cleaned_word_tokens]

# Plotting Data
data = pd.DataFrame(cleaned_words, columns=['word_tokens'])
data =
data.word_tokens.value_counts().reset_index().rename(columns={'word_token
s': 'counts', 'index': 'word_tokens'})
data = data.reset_index().rename(columns={'index': 'rank'})
data['rank'] = data['rank'] + 1

plot_data_top50_words = data.head(50)
plot_data_top20_words = data.head(50)

total = plot_data_top50_words[:50].counts.sum()

plt.tight_layout()
sns.barplot(data=plot_data_top50_words, x='word_tokens', y='counts')

rcParams['figure.figsize'] = 16, 8
plt.xticks(rotation=90)
plt.savefig('plot_data_top50_words.png')
plt.show()

alpha = 2 # zipf distribution parameter
total = plot_data_top50_words[:50].counts.sum()

plt.plot(range(50), [zipf.pmf(p, alpha) * total for p in range(1, 51)],
color='crimson', lw=3)
plt.tight_layout()
sns.barplot(data=plot_data_top50_words, x='word_tokens', y='counts')

rcParams['figure.figsize'] = 16, 8
plt.xticks(rotation=90)
plt.savefig('plot_data_top50_words_zipf.png')
plt.show()
```





2 Python programming. Text Similarity (20 points)

```
import re
import math
import pandas as pd
import string
from string import digits
from nltk import word_tokenize
import numpy as np
import nltk

def cleaning_data(data):
    textdata = data
    punctuation_free_txt = textdata.translate(str.maketrans("",string.punctuation))
    #making it lowercase
    lowercase_txt = punctuation_free_txt.lower()
    #removing digits
    digits_free_txt = lowercase_txt.translate(str.maketrans("", "", digits))
    text = list(digits_free_txt)
    text0_tokenized = word_tokenize(digits_free_txt)
    return text0_tokenized
```

```
f = open('Text_0.txt', 'r',errors='ignore')
data = f.read()

text0_tokenized = cleaning_data(data)

f = open('Text_1.txt', 'r',errors='ignore')
data = f.read()

text1_tokenized = cleaning_data(data)

f = open('Text_2.txt', 'r',errors='ignore')
data = f.read()

text2_tokenized = cleaning_data(data)

#print(text0)

#For complete and clean string

def making_string(text):
    completeString = ''
    for values in text:
        completeString += values + ' '
    return completeString


#print(text0_tokenized)

text0 = making_string(text0_tokenized)
text1 = making_string(text1_tokenized)
text2 = making_string(text2_tokenized)

line0 = re.sub('[!@#$â€™œ]', '', text0)
line1 = re.sub('[!@#$â€™œ]', '', text1)
line2 = re.sub('[!@#$â€™œ]', '', text2)

print(line0)
def calculateCosine(line1,line2):
```

```
    X_list = word_tokenize(line1)
    Y_list = word_tokenize(line2)

    X_set = set(X_list)
    Y_set = set(Y_list)
    rvector = X_set.union(Y_set)
    return rvector,X_set,Y_set

def createVectors(vectors,X_set,Y_set):
    l1 =[];l2 =[]
    for w in vectors:
        if w in X_set: l1.append(1) # create a vector
        else: l1.append(0)
        if w in Y_set: l2.append(1)
        else: l2.append(0)
    return l1,l2

def cosineFormula(line1,line2):
    vectors,X_set,Y_set = calculateCosine(line1,line2)
    l1,l2 = createVectors(vectors,X_set,Y_set)
    c = 0
    for i in range(len(vectors)):
            c+= l1[i]*l2[i]
    cosine = c / float((sum(l1)*sum(l2))**0.5)
    print("Cosine: ", cosine)




#function for finding the jaccard score of two texts
def jaccard_score(text, text1):
    intersection = len(set(text).intersection(set(text1)))
    union = (len(text) + len(text1)) - intersection
    return (intersection / union)


#for finding eucaledian distnace between all texts
def eucaledian_distance(txt1, txt2):
    return (np.sqrt(sum((txt1-txt2)**2)))
```

```
#MAKING VECTORS
def make_vectors(text0, text1):
    d1 = dict()
    d2 = dict()
    text_array = [text0, text1]
    for file in text_array:
        for word in file:
            if (word in text0) and (word in text1):
                d1[word] = 1
                d2[word] = 1
            elif (word in text0) and (word not in text1):
                d1[word] = 1
                d2[word] = 0
            elif (word not in text0) and (word in text1):
                d1[word] = 0
                d2[word] = 1
            else:
                d1[word] = 0
                d2[word] = 0

        vector1 = np.array(list(d1.values()))
        vector2 = np.array(list(d2.values()))
        return vector1, vector2


#finding jaccard scores of all texts
print('Jaccard Score for Txt0 and Txt1',jaccard_score(line0, line1))
print('Jaccard Score for Txt0 and Txt2',jaccard_score(line0, line1))
print('Jaccard Score for Txt2 and Txt1',jaccard_score(line1, line2))


#finding eucaledian_distance of all texts
vector0, vector1 = make_vectors(text0_tokenized, text01_tokenized)
print('\neucaledian_distance for txt0 and txt1 is ', eucaledian_distance(vector0, vector1))
vector0, vector2 = make_vectors(text0_tokenized, text02_tokenized)
print('eucaledian_distance for txt0 and txt2 is ', eucaledian_distance(vector0, vector2))
vector1, vector2 = make_vectors(text01_tokenized, text02_tokenized)
print('eucaledian_distance for txt1 and txt2 is ', eucaledian_distance(vector1, vector2))
```

cosineFormula(line0,line1)

cosineFormula(line0,line2)

cosineFormula(line1,line2)

2.1

Value of jaccard score is not 1 for any combination of Text_0, Text_1 and Text_2 and they are very much close to 0 which means they are not identical to each other.

According to eculidean distance score comparing text_0 and text_2 are found to be identical whereas comparing text_0 text_1 and text_1 and text_2 are not identical.

The value of cosine similarity of Text_0, Text_1, Text_2 shows that the all the three text files are not very similar to each other. There are multiple words which came similar in all three files but they are not very much similar in whole.

3 Short questions: (12 points)

3.1

The advantage of using the log-log plot is that every data points are visualized in one magnitude of intervals. It will show the rate of change occurs on each data points with the constants numbers of multiplying to go from one scale to another. We need to use the log-log plot in a scenario, where the ranges of the data are too high and difficult to visualize or interpret the data on the plots. We need to avoid the log-log plot if the data range have very low magnitude differences.

3.2

Cosine distance is advantageous to find the document similarity than the Euclidean distance in respective of their size, because it will go for the angel between the words rather the actual distance like in a Euclidean distance. For example, if one word comes up to 100 times in one document and 60 times in other document then it will find the angel between them instead of the distance apart. So, smaller the angel more similar the documents will be.

3.3

The Laplacian, aka +1 smoothing is a technique used to ignore the zero probabilities in classifying the datasets. It will help us to give some sort of the normalized probabilities on each word to find the similarities in a document.