Lab3_Problem2

February 22, 2020

[ ]:

```python
# Extract pdfs from website

import os
import requests
from urllib.parse import urljoin
from bs4 import BeautifulSoup

url = "http://proceedings.mlr.press/v70/"

# If the folder does not exist, create one automatically
folder_location = './webscraping/'
if not os.path.exists(folder_location):
    os.mkdir(folder_location)

response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")
for link in soup.select("a[href$='.pdf']"):
    #Name the pdf files using the last portion of each link
    filename = os.path.join(folder_location, link['href'].split('/')[-1])
    with open(filename, 'wb') as f:
        f.write(requests.get(urljoin(url, link['href'])).content)
```

[145]:

```python
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage
from io import StringIO
import os

# Convert a given pdf to text and return the text
def convert_pdf_to_txt(pathname):
    rsrcmgr = PDFResourceManager()
    retstr = StringIO()
    codec = 'utf-8'
    laparams = LAParams()
    device = TextConverter(rsrcmgr, retstr, codec=codec, laparams=laparams)
    fptr = open(pathname, 'rb')
    interpreter = PDFPageInterpreter(rsrcmgr, device)

    try:
        for page in PDFPage.get_pages(fptr, set(), maxpages=0,␣
 ↪password="",caching=True, check_extractable=True):
            interpreter.process_page(page)
    except: # Need this for an exception that gets thrown for pdfs that can't␣
 ↪be converted
        return ""

    text = retstr.getvalue()

    fptr.close()
    device.close()
    retstr.close()
    return text

# Iterate through files in a given directory
# Convert the file to text and then store in a list
def store_text_to_list(path_name, file_extension):
    converted_text_list = []
    count = 0 # Keep track of the number of files converted

    for filename in os.listdir(path_name):
        if filename.endswith(file_extension):
            converted_text = convert_pdf_to_txt(path_name + filename)
            if(converted_text != ""):
                converted_text_list.append(converted_text)
                count += 1

    print("Number of files converted:",count)
    return converted_text_list
```

[146]:

```
# Convert all pdfs in the directory to text
converted_text_list = store_text_to_list('./pdfs/', '.pdf')
```

```
Number of files converted: 720
```

[ ]:

```
import string
from nltk.corpus import words
import nltk

nltk.download('words')

# Iterate through list and create a dictionary with (key, value) pairs being␣
 ↪(word, frequency)
```

```
# Add a word to dictionary if it's not already there, else update the entry by 1
freq_dict = {}


for document in converted_text_list:
    if(document != ""):
        split_document = document.split()
        for word in split_document:
            if word in words.words():
                if word in freq_dict:
                    freq_dict[word] = freq_dict[word] + 1
                else:
                    freq_dict[word] = 1
```

```
[nltk_data] Downloading package words to /Users/musarafik/nltk_data…
[nltk_data]   Unzipping corpora/words.zip.
```

[ ]:

```
# Sort dictionary in reverse order and create a list of tuples so we can easily␣
 ↪create a dataframe
sorted_d = sorted(((value, key) for (key,value) in freq_dict.items()),␣
 ↪reverse=True)
```

[1]:

```
import pickle
# Use pickle to save dictionary so we don't have to keep converting pdfs to␣
 ↪text everytime we restart notebook

# WARNING: do not uncomment and run this or else b will be loaded with junk
# with open('filename.pickle', 'wb') as handle:
#     pickle.dump(sorted_d, handle, protocol=pickle.HIGHEST_PROTOCOL)

# with open('freq_dict.pickle', 'wb') as handle:
#     pickle.dump(freq_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)

with open('freq_dict.pickle', 'rb') as handle:
    saved_dict = pickle.load(handle)

# b holds all the list of words and their frequencies
with open('filename.pickle', 'rb') as handle:
    b = pickle.load(handle)
```

[104]:

```
import pandas as pd

clean_dict = sorted(((value, key) for (key, value) in saved_dict.items()),␣
 ↪reverse=True)

# Convert list of tuples to dataframe and display top 15 words/symbols
df = pd.DataFrame(b, columns = ['Frequency', 'Word'])
```

```python
df.head(15)
```

```
[104]:      Frequency  Word
        0      184869   the
        1      101466    of
        2       86891   and
        3       61703    to
        4       57986     a
        5       55017    is
        6       50851    in
        7       47783     =
        8       44202   for
        9       36039  that
        10      34678    we
        11      29098  with
        12      27959     1
        13      26949     -
        14      24756     +
```

As we can see from the dataframe, the top ten most frequent words are: 1. 'the' - 184869 occurrences 2. 'of' - 101466 occurrences 3. 'and' - 86891 occurrences 4. 'to' - 61703 occurrences 5. 'a' - 57986 occurrences 6. 'is' - 55017 occurrences 7. 'in' - 50851 occurrences 8. 'for' - 44202 occurrences 9. 'that'- 36039 occurrences 10. 'we' - 34678 occurrences

```python
[105]: # Add column of probabilities for each word
       totalWords = df['Frequency'].sum()
       df['Probability'] = df['Frequency'].divide(totalWords)

       df.head(15)
```

```
[105]:      Frequency  Word  Probability
        0      184869   the     0.042043
        1      101466    of     0.023076
        2       86891   and     0.019761
        3       61703    to     0.014033
        4       57986     a     0.013187
        5       55017    is     0.012512
        6       50851    in     0.011565
        7       47783     =     0.010867
        8       44202   for     0.010053
        9       36039  that     0.008196
        10      34678    we     0.007887
        11      29098  with     0.006618
        12      27959     1     0.006359
        13      26949     -     0.006129
        14      24756     +     0.005630
```

[106]:
```
from scipy.stats import entropy

# Calculate entropy:
entropy(df['Probability'])
```

[106]:
```
8.416007866175365
```

By using Scipy, we calculated the entropy to be 8.416.

[107]:
```
import re
import numpy as np
from numpy.random import Generator, PCG64

# Clean some of the symbols
df['Word'].str.replace('[^a-zA-Z]', '')

# Random number generator
rg = Generator(PCG64())

words = np.array(dfClean['Word'])
probabilities = np.array(df['Probability'])

wordList = []
for word in words:
    wordList.append(word)

probList = []
for prob in probabilities:
    probList.append(prob)

# Create a 10-sentence paragraph with a random number of words for each sentence
# sampled out of our distribution using np.random.choice()
paragraph = ""
for i in range(10):
    sentence = ""
    x = rg.integers(20)
    for j in range(x):
        #sentence += np.random.choice(wordList, 1, True, probList) + " "
        temp = np.random.choice(wordList, 1, True, probList)
        temp = np.array2string(temp)
        sentence += temp + " "
    paragraph += sentence +  "."
```

[110]:
```
paragraph = paragraph.replace("[", "")
paragraph = paragraph.replace("]", "")
paragraph = paragraph.replace("'", "")
paragraph = paragraph.replace("(", "")
paragraph = paragraph.replace(")", "")
```

[111]:

```
print(paragraph)
```

```
marginal Lists   Rb.  · in al., T convex · harmless related architecture = the
and .Let hamper for to in .Indeed, from given range + the Trek: 222-230, =
.Bayesian each is 000 ˆF Maclaurin, added ferentiable log-likelihood  maxy:
LUCB-G F our length maximums .over 224 bound 0.07 Fig. solution was condition
and of with .5.1. min   bounded if of observing   show neural during f .  Tom,
and is resulting least hidden and express zi xr Proposition .. Tong 319 in and
detail:  rst-order ity the ference synthetic Zhao, 2009 and want .y* Jiang,
where Josip, work   any round Harrison side are theory the least for for .N
operator Acknowledgements which 2, word-by-word. large likelihood cid:110 .
```

Our synthesized paragraph is:

marginal Lists Rb. · in al., T convex · harmless related architecture = the and .Let hamper for to in .Indeed, from given range + the Trek: 222–230, = .Bayesian each is 000 ˆF Maclaurin, added ferentiable log-likelihood maxy: LUCB-G F our length maximums .over 224 bound 0.07 Fig. solution was condition and of with .5.1. min bounded if of observing show neural during f . Tom, and is resulting least hidden and express zi xr Proposition .. Tong 319 in and detail: first-order ity the ference synthetic Zhao, 2009 and want .y∗ Jiang, where Josip, work any round Harrison side are theory the least for for .N operator Acknowledgements which 2, word-by-word. large likelihood cid:110 .

[ ]:

```

```