text_rng

August 21, 2021

[1]:
```
import sys
import os

sys.path.insert(0, os.getcwd() + '/reddit_download')
```

[2]:
```
import numpy as np
import matplotlib.pyplot as plt

sys.path.append('../..')
from plotting.matplotlib_setup import configure_latex, savefig,␣
 ↪set_size_decorator, savefig, thiner_border

tex_dir, images_dir = 'porocilo/main.tex', 'porocilo/images'

configure_latex(style=['science', 'notebook'], global_save_path=images_dir)

%config InlineBackend.figure_format = 'pdf'
```

0.1 preprocess and make csv

[3]:
```
# from reddit_download.RWV.pushshift.utils import build_df

# df_comments = build_df(content_type='comment', file_path=os.getcwd() + '/
 ↪reddit_download')
# df_posts = build_df(content_type='post', file_path=os.getcwd() + '/
 ↪reddit_download')

# ind = df_comments[df_comments['author'] == '[deleted]'].index
# df_comments.drop(ind, inplace=True)

# ind = df_comments[df_comments['author'] == 'AutoModerator'].index
# df_comments.drop(ind, inplace=True)

# ind = df_posts[df_posts['author'] == '[deleted]'].index
# df_posts.drop(ind, inplace=True)

# ind = df_posts[df_posts['author'] == 'AutoModerator'].index
# df_posts.drop(ind, inplace=True)

# df_comments = df_comments.rename(columns={"link_id": "post_id"})

# df_comments = df_comments.rename(columns={"created_utc": "timestamp"})
# df_posts = df_posts.rename(columns={"created_utc": "timestamp"})

# df_comments.to_csv('comments.csv', index=False)
# df_posts.to_csv('posts.csv', index=False)
```

0.2 modin and ray stuff

[4]:

```
# import pandas as pd
# import swifter

os.environ["MODIN_ENGINE"] = "ray"
os.environ["MODIN_CPUS"] = "8"
import ray
ray.init(num_cpus=8)
import modin.pandas as pd

#import swifter

# from distributed import Client
# client = Client()

# workers = 12

# os.environ["MODIN_ENGINE"] = "ray"
# os.environ["MODIN_CPUS"] = str(workers)

# import ray
# ray.init(num_cpus=workers)

# import modin.pandas as pd

from tqdm import tqdm
from modin.config import ProgressBar
ProgressBar.enable()
```

```
2021-08-21 19:20:07,472 INFO services.py:1245 -- View the Ray dashboard at
http://127.0.0.1:8265
```

[5]:

```
df_comments = pd.read_csv('comments.csv', lineterminator='\n')
df_posts = pd.read_csv('posts.csv', lineterminator='\n')
```

[6]:

```
df_comments.drop(columns=['author', 'timestamp', 'post_id', 'parent_id',␣
 ↪'permalink'], inplace=True)
df_posts.drop(columns=['author', 'timestamp', 'post_id', 'num_comments',␣
 ↪'permalink'], inplace=True)
```

[7]:

```
df_comments['body'] = df_comments['body'].apply(lambda x: str(x))
```

0.3 make sentences with NLTK tokenizer

[8]:

```
from reddit_download.RWV.text_processing.process_reddit import word2vec_input
```

```
Estimated completion of line 1:   0%          Elapsed time: 00:00, estimated␣
 ↪remaining time: ?
```

[9]:

```
class TokenizerInput:
    def __init__(self, text):
        self.body = str(text)
        self.is_post = False

def body_to_sent(x):
    return word2vec_input([TokenizerInput(x)], to_sent=True)
```

[10]:

```
# df_comments['sent'] = df_comments['body'].apply(body_to_sent)
```

0.4 sentence count

[11]:

```
# counts = df_comments['sent'].apply(len).values
```

[12]:

```
# fig, ax = set_size_decorator(plt.subplots, fraction=0.5, ratio='4:3')(1, 1)

# ax.hist(counts, bins=14, range=(1, 15), histtype='step')
# ax.set_xlabel(r'\# stavkov')
# ax.set_ylabel(r'$N$')
# savefig('sent_count', tight_layout=False)
```

0.5 char in body count

[13]:

```
# char_counts = df_comments['body'].apply(len).values
```

[14]:

```
# fig, ax = set_size_decorator(plt.subplots, fraction=0.5, ratio='4:3')(1, 1)

# plt.hist(char_counts, range=(0, 1000), bins=100, histtype='step')
# ax.set_xlabel(r'\# znakov v komentarju')
# ax.set_ylabel(r'$N$')
# savefig('char_comment_counts', tight_layout=False)
```

0.6 char in sent count

[15]:

```
class SentCharCounter:
    def __init__(self):
        self.counts = []

    def count(self, sent_lst):
        for s in sent_lst:
            self.counts.append(len(s))
        return self
```

[16]:

```
# SC = SentCharCounter()

# sent_char_counts = df_comments['sent'].apply(SC.count)
```

[17]:

```
# counts = sent_char_counts[0].counts
```

[18]:

```
# fig, ax = set_size_decorator(plt.subplots, fraction=0.5, ratio='4:3')(1, 1)

# ax.hist(counts, range=(0, 500), bins=100, histtype='step')
# ax.ticklabel_format(style='sci', axis='y', scilimits=(0, 0))
# ax.set_xlabel(r'\# znakov v stavku')
# ax.set_ylabel(r'$N$')
# savefig('sent_word_count', tight_layout=False)
```

0.7 unique word count

[19]:

```
from collections import Counter

class WordCounter:
    def __init__(self):
        self.dct = dict()

    def count_words(self, s):
        count = dict(Counter(s.split()))
        for k, v in count.items():
            if k not in self.dct:
                self.dct[k] = v
            else:
                self.dct[k] += v

        return self
```

[20]:

```
# WC = WordCounter()

# res = df_comments['body'].swifter.apply(WC.count_words)
```

[21]:
```
# word_dct = res[0].dct
```

[22]:
```
# sorted_word_dct = {k: v for k, v in sorted(word_dct.items(), key=lambda item:␣
 ↪item[1], reverse=True)}
```

[23]:
```
# wv = list(sorted_word_dct.values())[:50]
# wk = list(sorted_word_dct.keys())[:50]
```

[24]:
```
# fig, ax = set_size_decorator(plt.subplots, fraction=0.5, ratio='4:3')(1, 1)

# ax.bar(wk, wv)
# plt.xticks(rotation=90, fontsize=5)
# ax.minorticks_off()
# ax.ticklabel_format(style='sci', axis='y', scilimits=(0, 0))
# savefig('word_count', tight_layout=False)
```

[25]:
```
df_comments = df_comments._to_pandas()
ray.shutdown()
```

0.8 bitstream for RNG

[26]:
```
from benford_helper_functions import str_to_bits
```

[27]:
```

```

[28]:

```
df_comments.sort_values(by=['score'], inplace=True, ascending=False)
```

[29]:

```
# def text_to_bitstream(text_lst, max_bits=10**6):
#     bit_streams = [[] for i in range(8)]
#     for count, text in enumerate(text_lst):
#         bits = str_to_bits(text, one_byte=False, remove_spaces=True,␣
↪to_replace=top_1000_words[:256])
#         bits_lst = bits.split(" ")

#         for byte in bits_lst:
#             for i, b in enumerate(byte.zfill(8)):
#                 bit_streams[i].append(b)

#         bit_count = len(bit_streams[0])

#         if count % 5000 == 0:
#             print(bit_count / max_bits * 100)

#         if bit_count > max_bits:
#             return bit_streams, count
#     return bit_streams
```

[30]:
```
# bit_streams, count = text_to_bitstream(df_comments['body'].values,␣
 ↪max_bits=100 * 10**6)
```

[31]:
```
from NIST_tests import RNG_test
```

[32]:
```
# test_n_bit_streams = 1
# max_bits = 10**6

# results = []
# for c, bit_stream in enumerate(bit_streams[2:]):
#     print(c)
#     bits = ''.join(bit_stream)

#     bit_pos_results = []
#     for i in range(test_n_bit_streams):
#         test_bits = bits[i*max_bits:max_bits]
#         res = RNG_test(test_bits)
#         bit_pos_results.append(res)

#     results.append(bit_pos_results)
```

0.9 LCG

[33]:
```
from random_helper_functions import bin_str_to_matrix, split_to_arr
```

[34]:
```
# r = ''.join(bit_streams[7])
```

[35]:
```
# rr = r[10**6:2*10**6]
```

[36]:
```
# RNG_test(rr)
```

[37]:
```
# m = bin_str_to_matrix(split_to_arr(rr))
```

[38]:
```
def make_LCG_bits(bits, n=32, num_bits=10**6, a=48271, c=0, mod=2**32, k=0,␣
 ↪no_chunked=True):
    m = len(bits) // n

    bits_chunked = [bits[i*m:(i+1)*m] for i in range(n)]

    new_bits = ''
    for i in range(m):

        if no_chunked:
            b = bits[i*n:(i+1)*n]
        else:
            b = ''
            for j in range(n):
                b += bits_chunked[j][i]

        if mod != 0:
            b = bin((int(b, 2) * a + c) % mod)[2:]
        else:
            b = bin(int(b, 2) * a + c)[2:]

        if k != 0:
            new_bits += b[int(len(b) - len(b) * k):]
        else:
            new_bits += b[len(b)//2:]

        if len(new_bits) > num_bits:
            return new_bits, i * n

    return new_bits
```

[39]:

```
# bit_str = ''.join(bit_streams[7])
# st, used = make_LCG_bits(bit_str, num_bits=10**6,
#                           a=1664525, c=1013904223, mod=2**32 - 1, k=0, n=32)
```

[40]:

```
# RNG_test(st)
```

[41]:

```
# used / 10**6
```

0.10 chunks

[42]:

```
def make_bit_chunk(bits, n):
    m = len(bits) // n
    bits_chunked = [bits[i*m:(i+1)*m] for i in range(n)]
    return bits_chunked


def make_bit_chunks(bits, n=32, splits=2, prnt=False):
    end_parts, elements = n**(splits + 1), len(bits) // n**(splits + 1)
    if prnt:
        print(f'end parts: {end_parts} with {elements} elements')
    bits_chunked = make_bit_chunk(bits, n)

    if splits == 0:
        return bits_chunked, end_parts, elements

    for split in range(splits):
        split_chunks = []
        for chunk in bits_chunked:
            split_chunks += make_bit_chunk(chunk, n)
        bits_chunked = split_chunks

    return bits_chunked, end_parts, elements


def make_bitstring_from_chunks(bits, num_bits=10**6, **kwargs):
    bits_chunked, n_chunks, elements = make_bit_chunks(bits, **kwargs)

    bitstring = ''
    for i in range(elements):
        for j in range(n_chunks):
            b = bits_chunked[j][i]
            bitstring += b
            if len(bitstring) > num_bits:
                return bitstring

    return bitstring
```

[43]:

```
#st = np.arange(0, 12, 1).astype(str)
st = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l']
st = ''.join(st).upper()
```

[44]:

```
def multi_mix(st, n_mixes=None, chunks=None):
    starting_st = st

    if chunks is None:
        n = int(np.sqrt(len(st))) - 1
    else:
        n = chunks
    print(f'splits: {n}')

    if n_mixes is None:
        n_mixes = n

    for i in tqdm(range(n_mixes)):
        st = make_bitstring_from_chunks(st, n=n, splits=1)
        if st == starting_st:
            print('sequence repeated! returnig last good combination!')
            return old_st
        old_st = st

    return st
```

[45]:

```
multi_mix(st, chunks=3)
```

```
splits: 3
```

```
100%|
                    | 3/3 [00:00<00:00, 30840.47it/s]
```

[45]:
```
'ABCEFGIJK'
```

[46]:
```
# c = make_bitstring_from_chunks(bit_str, num_bits=10**6, n=32, splits=2)
```

[47]:
```
text = df_comments['body']
```

[48]:
```
full_text = ''.join(text)[:10**6]
```

[49]:
```
spaces_bits = str_to_bits(full_text, to_replace=top_1000_words[:100],␣
 ↪remove_spaces=True)
```

[50]:
```
list_bits = list(spaces_bits.split(" "))

last_bit = ''
for b in list_bits:
    last_bit += b[-1]
```

[51]:
```
# RNG_test(last_bit[:2*10**6][::2])
```

[52]:
```
# mm = multi_mix(last_bit[:1*10**6], n_mixes=10)
```

[53]:
```
# RNG_test(mm)
```

0.11 diag

[54]:
```
def valid_shapes(num):
    shapes = []
    lim = int(np.sqrt(num))
    for i in range (1, lim):
        if num % i == 0:
            shapes.append([i, int(num/i)])

    return shapes[::-1]
```

[55]:
```
import itertools

def diag_rng(bit_arr, reverse_shapes=False, reverse_sort=False):
    if reverse_shapes:
        shapes = valid_shapes(len(bit_arr))[:-1][::-1]
    else:
        shapes = valid_shapes(len(bit_arr))[:-1]
    for shape in tqdm(shapes):
        new_bit_arr = bit_arr.reshape(shape[0], shape[1])

        m = max(shape)
        r = np.arange(-m, m + 1, 1)

        new_s = []
        for i in r:
            s = np.diag(new_bit_arr, k=i).astype(str)
            if len(s) != 0:
                new_s.append(''.join(s))

        new_s.sort(key=lambda x: len(x[0]), reverse=reverse_sort)
        new_s = list(itertools.chain.from_iterable(new_s))
        new_s = ''.join(new_s)

        bit_arr = split_to_arr(new_s)

    return new_s
```

[57]:

```
a = last_bit[:10**6]

# a = diag_rng(split_to_arr(a), reverse=False)
# a = make_bitstring_from_chunks(a, num_bits=1*10**6, n=32, splits=0)
diag_bits = diag_rng(split_to_arr(a))
mm = multi_mix(diag_bits, n_mixes=1, chunks=32)
# diag_bits = diag_rng(split_to_arr(mm))
```

```
100%|
                   | 11/11 [00:06<00:00,  1.79it/s]

splits: 32

100%|
                    | 1/1 [00:00<00:00, 19.17it/s]
```

[58]:

```
RNG_test(mm)
```

```
100%|
                   | 16/16 [00:03<00:00,  4.53it/s]
```

[58]:

```
                                          test     p
0                 Frequency Test (Monobit)  0.00
1            Frequency Test within a Block  0.53
2                                 Run Test  0.00
3           Longest Run of Ones in a Block  0.00
4                   Binary Matrix Rank Test  0.03
5   Discrete Fourier Transform (Spectral) Test  0.04
```

```
6       Non-Overlapping Template Matching Test  0.00
7           Overlapping Template Matching Test  0.03
8          Maurer's Universal Statistical test  0.04
9                       Linear Complexity Test  0.43
10                                 Serial test  0.51
11                    Approximate Entropy Test  0.03
12             Cummulative Sums (Forward) Test  0.00
13                      Random Excursions Test  0.83
14              Random Excursions Variant Test  0.86
```

[59]:

```
a = last_bit[:1*10**6]
a = make_bitstring_from_chunks(a, num_bits=1*10**6, n=32, splits=2)
a_arr = split_to_arr(a)
```

[60]:

```
diag_bits = diag_rng(a_arr)
```

```
100%|
                   | 17/17 [00:08<00:00,  2.10it/s]
```

[61]:

```
RNG_test(diag_bits)
```

```
100%|
                   | 16/16 [00:02<00:00,  5.57it/s]
```

[61]:

```
                                          test     p
0                     Frequency Test (Monobit)  0.00
1                Frequency Test within a Block  0.60
2                                     Run Test  0.00
3               Longest Run of Ones in a Block  0.00
4                      Binary Matrix Rank Test  0.68
5   Discrete Fourier Transform (Spectral) Test  0.45
6       Non-Overlapping Template Matching Test  0.01
7           Overlapping Template Matching Test  0.01
8          Maurer's Universal Statistical test  0.06
9                       Linear Complexity Test  0.05
10                                 Serial test  0.56
11                    Approximate Entropy Test  0.01
12             Cummulative Sums (Forward) Test  0.00
13                      Random Excursions Test  0.16
14              Random Excursions Variant Test  0.80
```

[62]:

```
RNG_test(a[::2])
```

```
100%|
                   | 16/16 [00:01<00:00, 11.07it/s]
```

[62]:

```
                                          test     p
0                     Frequency Test (Monobit)  0.00
1                Frequency Test within a Block  0.61
2                                     Run Test  0.00
3               Longest Run of Ones in a Block  0.32
4                      Binary Matrix Rank Test  0.83
5   Discrete Fourier Transform (Spectral) Test  0.02
6       Non-Overlapping Template Matching Test  0.27
7           Overlapping Template Matching Test  0.07
8          Maurer's Universal Statistical test   nan
9                       Linear Complexity Test  0.97
10                                 Serial test  0.11
11                   Approximate Entropy Test  0.07
12             Cummulative Sums (Forward) Test  0.00
13                     Random Excursions Test  0.25
14             Random Excursions Variant Test  0.25
```

[63]:

```python
# from bitstring import BitArray

# def float_from_bitstring(bitstring):
#     return BitArray(bin=bitstring).float
```

[72]:

```python
def make_ints_with_n_bits(bits, n):
    m = len(bits) // n

    ints = []
    z = 0
    for i in range(m):
        take = bits[i*n:(i+1)*n]
        make_int = int(take, 2)
        if make_int != 0:
            ints.append(make_int)
        else:
            z += 1

    print(f'{z} total zeros')
    return np.array(ints)
```

[65]:

```python
def reshape_and_truncate(arr, shape):
    desired_size_factor = np.prod([n for n in shape if n != -1])
    if -1 in shape:  # implicit array size
        desired_size = arr.size // desired_size_factor * desired_size_factor
    else:
        desired_size = desired_size_factor
    return arr.flat[:desired_size].reshape(shape)
```

[69]:

```python
def text_lognormal_dist(bits, n, d):
    """
    bits: str
        Sequence of bits
    n: int
        Number of bits to take together in bits sequence
    d: int
        Number of multiplications
    """
    ints = make_ints_with_n_bits(last_bit, n=n)
    ints_mat = reshape_and_truncate(bits, (len(ints) // d, d))
    ints_prod = np.prod(ints_mat, axis=1).astype(np.float32)
    return ints_prod
```

[ ]:

```

```

[ ]:

```

```