new_text_rng

August 21, 2021

[1]:
```
import sys
import os

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

[2]:
```
import numpy as np
import pandas as pd
from tqdm import tqdm
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 load data

[3]:
```
df_comments = pd.read_csv('comments.csv', lineterminator='\n')
df_posts = pd.read_csv('posts.csv', lineterminator='\n')

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

[4]:
```
df_comments['body'] = df_comments['body'].apply(lambda x: str(x))
```

[5]:
```
df_comments.sort_values(by=['timestamp'], inplace=True, ascending=False)
```

1 RNG

[6]:
```
from benford_helper_functions import str_to_bits, get_bitstring,␣
↪binary_tree_walk
from random_helper_functions import split_to_arr, bin_str_to_matrix
from NIST_tests import RNG_test
```

[7]:
```
# r/Genshin_Impact leaking much
```

[8]:
```
from text_rng import TextRng
```

[9]:
```
# df_comments.sort_values(by=['score'], inplace=True, ascending=False)
```

[10]:
```
# ocena bitov/s
TR = TextRng(text=df_comments['body'].values[:10**6],
             utf8_kwargs={"utf8_bit_pos": -1, "top_words": top_1000_words[:
 ↪100], "remove_spaces": True},
             mixing_kwargs = {"n_mixes": 1, "chunks": 16},
             lognormal_kwargs = {"n": 8, "d": 6, "div": 1e6},
             bit_generation="bitstring")
```

[11]:
```
from benford_helper_functions import str_to_bits
```

[12]:
```
all_bits = str_to_bits(TR.text, remove_spaces=False)
```

[13]:
```
f'{len(all_bits):.3e}'
```

[13]:
```
'9.321e+08'
```

[14]:
```
bits_per_comment = len(all_bits) / 10**6
bits_per_comment
```

[14]:
```
932.128281
```

[15]:
```
bits = TR.run()
```

[16]:
```
r = len(all_bits) / len(bits)
r
```

[16]:
```
38.85320947534378
```

[17]:
```
(bits_per_comment / r) * 100
```

[17]:
```
2399.1023999999998
```

1.1 testing code

[18]:
```
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=None, **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 num_bits:
                if len(bitstring) > num_bits:
                    return bitstring

    return bitstring


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'chunks: {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=0)
        if st == starting_st:
            print('sequence repeated! returnig last good combination!')
            return old_st
        old_st = st

    return st
```

[19]:
```
def utf8_bits(text, utf8_bit_pos=-1, n_top_word_replace=100):
    full_text = ''.join(text)
    spaces_bits = str_to_bits(full_text, to_replace=top_1000_words[:
 ↪n_top_word_replace], remove_spaces=True)

    list_bits = list(spaces_bits.split(" "))

    bits = ''
    for b in list_bits:
        try:
            bits += b[utf8_bit_pos]
        except Exception as e:
            print(e, b)
            pass
    return bits
```

[20]:
```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)


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)


def text_lognormal_dist(bits, n, d, div=1):
    """
    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(bits, n=n)
    ints_mat = reshape_and_truncate(ints, (len(ints) // d, d))
    ints_prod = np.prod(ints_mat / div, axis=1)
    return ints_prod
```

[21]:
```python
def make_float_chunks(fl, n):
    m = len(fl) // n
    bits_chunked = [fl[i*m:(i+1)*m] for i in range(n)]
    return bits_chunked

def make_floatarr_from_chunks(fl, num_fl=None, n=2): # n -> chunks
    floats_chunked = make_float_chunks(fl, n)
```

```
    elements = len(fl) // n

    floatarr = []
    for i in range(elements):
        for j in range(n):
            f = floats_chunked[j][i]
            floatarr.append(f)

            if num_fl and len(floatarr) > num_fl:
                return floatarr

    return np.array(floatarr)

def multi_floatarr_from_chunks(fl, n_mixes, **kwargs):
    for m in tqdm(range(n_mixes)):
        fl = make_floatarr_from_chunks(fl, **kwargs)
    return np.array(fl)
```

[22]:
```
from stat_tests import chi2_test, ks_test


def bitstring_rng_test(rng_bits, take):
    it = len(rng_bits) // take

    if it < 1:
        it = 1

    results = []
    for i in range(it):
        print(i, '/', it)
        res = RNG_test(rng_bits[i*take:(i+1)*take])
        results.append(res)

    return results


def rng_all_comments_stat_tests(df, df_col, n_comments, bit_pos=-1, take=10**6,␣
 ↪use_walk=False):
    it = len(df[df_col]) // n_comments

    stat_results, rng_results = [], []

    for i in range(it):
        print(i, '/', it)
        comment_bits = utf8_bits(df_comments[df_col].values[i*n_comments:
 ↪(i+1)*n_comments], utf8_bit_pos=bit_pos)
        comment_bits = multi_mix(comment_bits, n_mixes=1, chunks=16)
```

```
        prod = text_lognormal_dist(comment_bits, n=8, d=6, div=1e6)

        u = np.log10(prod) % 1
        chi2, ks = chi2_test(u), ks_test(u)

        # rng_bits = get_bitstring(u)
        if use_walk:
            rng_bits = binary_tree_walk(u).astype(str)
        else:
            rng_bits = get_bitstring(u)

        print(f'NUM BITS: {len(rng_bits)}')
        rng_bits = "".join(rng_bits)

        res = bitstring_rng_test(rng_bits, take=take)

        stat_results.append([chi2, ks])
        rng_results.append(res)

    return stat_results, rng_results
```

[23]:
```
import pickle
```

[24]:
```
# results = rng_all_comments_stat_tests(df_comments, 'body', 10**6, bit_pos=-2)

# pickle.dump(results, open("results_bit_m2.p", "wb"))
# results = pickle.load(open("results_bit_m2.p", "rb"))
```

[25]:
```
# results = rng_all_comments_stat_tests(df_comments, 'body', 10**6, bit_pos=-3)

# pickle.dump(results, open("results_bit_m3.p", "wb"))
# results = pickle.load(open("results_bit_m3.p", "rb"))
```

[26]:
```
# results = rng_all_comments_stat_tests(df_comments, 'body', 10**6, bit_pos=-4)

# pickle.dump(results, open("results_bit_m4.p", "wb"))
# results = pickle.load(open("results_bit_m4.p", "rb"))
```

[27]:
```
# results = rng_all_comments_stat_tests(df_comments, 'body', 10**6, bit_pos=4,␣
 ↪use_walk=True)

# pickle.dump(results, open("results_bit_m5_walk.p", "wb"))
# results = pickle.load(open("results_bit_m5_walk.p", "rb"))
```

[28]:
```
res1 = pickle.load(open("results_bit_m1.p", "rb"))
res2 = pickle.load(open("results_bit_m2.p", "rb"))
res3 = pickle.load(open("results_bit_m3.p", "rb"))
res4 = pickle.load(open("results_bit_m4.p", "rb"))
res5 = pickle.load(open("results_bit_m5.p", "rb"))
```

[29]:
```
def get_p_results(results):
    p_results = []

    for r in results[1]:
        for ri in r:
            p_results.append(ri['p'].values.astype(np.float32))

    return np.array(p_results)


def get_stat_results(results):
    chi2, ks = [], []
    chi2_crit, ks_crit = [], []

    for r in results[0]:
        chi2.append(r[0][0][0][0])
        chi2_crit.append(r[0][1])
        ks.append(r[1][0][0][0])
        ks_crit.append(r[1][1])

    return chi2, ks, chi2_crit, ks_crit
```

[30]:
```
results = [res1, res2, res3, res4, res5]
results = results[:3]

p_results = [get_p_results(i) for i in results]
```

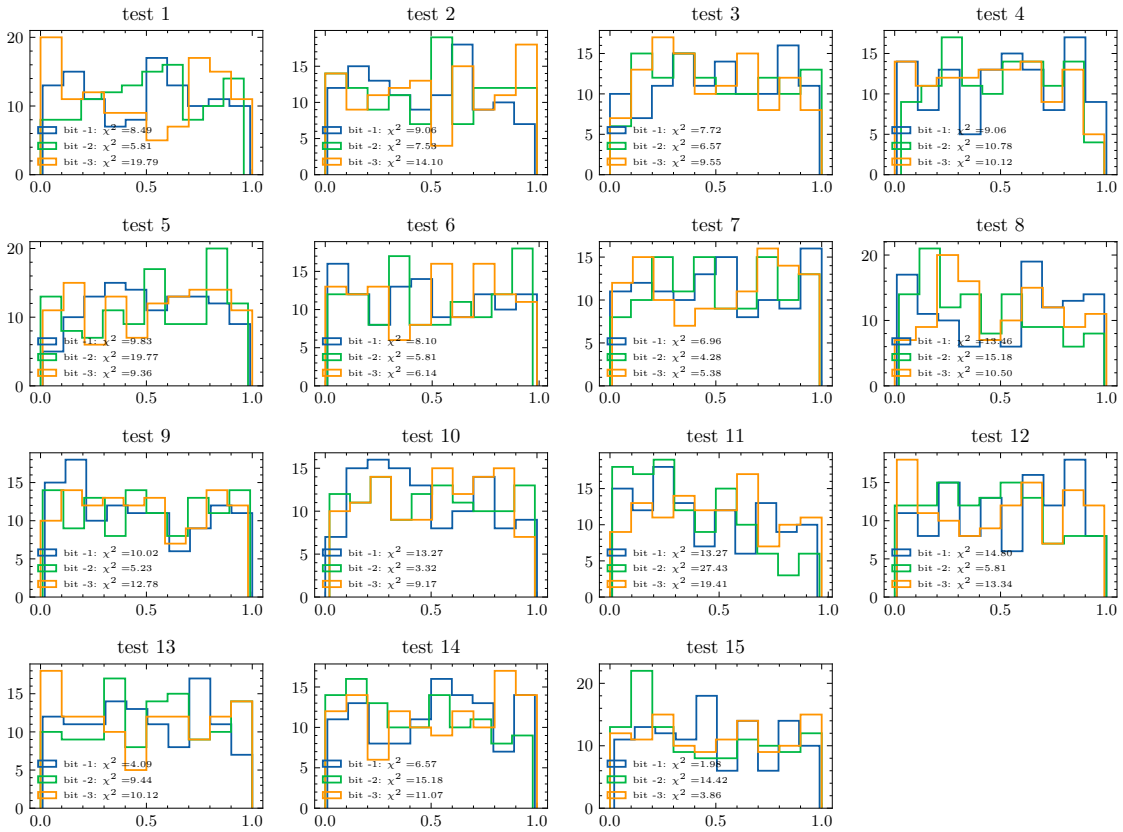
[31]:
```
fig, ax = set_size_decorator(plt.subplots, fraction=1.5, ratio='4:3')(4, 4)
ax[-1, -1].set_visible(False)
axs = ax.flatten()

# crit: 24.724970311318277
for i in range(15):
    l = []
    for j in range(len(p_results)):
        p = p_results[j]
        _, bins, _ = axs[i].hist(p[:, i], histtype='step', bins=10)
        c2 = chi2_test(p[:, i], n_bins=len(bins))
        l.append(f'bit {-j-1}: $\chi^2=${c2[0][0][0]:.2f}')

    axs[i].legend(l, loc='lower left', fontsize=5)
```

```
    axs[i].set_title(f'test {i+1}')

savefig('text_rng_p_dists')
```



[32]:
```
for i, res in enumerate(results):
    chi2, ks, chi2_crit, ks_crit = get_stat_results(res)
    print(chi2, chi2_crit)
    print()
```

```
[13.252473425060971, 12.539541436165774, 20.92652779948687, 4.865622720322977,
10.473310411227793, 17.29431565449117, 31.84311848291201, 9.090099880935266,
12.417349626675895] [23.209251158954356, 23.209251158954356, 23.209251158954356,
23.209251158954356, 23.209251158954356, 23.209251158954356, 23.209251158954356,
23.209251158954356, 23.209251158954356]

[8.40916154884962, 4.735473184971859, 6.64158861236574, 16.92785606860132,
24.235304444290886, 9.405207804278744, 5.765950267304573, 16.604384907216094,
5.257667204044474] [23.209251158954356, 23.209251158954356, 23.209251158954356,
23.209251158954356, 23.209251158954356, 23.209251158954356, 23.209251158954356,
23.209251158954356, 23.209251158954356]
```

```
[180.10312042812782, 265.3615221077625, 201.96343570990467, 215.2343119147305,
182.2467707337005, 160.09470273895766, 218.52919083556927, 299.86678973340946,
308.95789782570523] [23.209251158954356, 23.209251158954356, 23.209251158954356,
23.209251158954356, 23.209251158954356, 23.209251158954356, 23.209251158954356,
23.209251158954356, 23.209251158954356]
```

1.2 more testing

[33]:
```
n_comments = 10**6

comment_bits = utf8_bits(df_comments['body'].values[:n_comments])
```

[34]:
```
comment_bits = multi_mix(comment_bits, n_mixes=1, chunks=16)

prod = text_lognormal_dist(comment_bits, n=8, d=6, div=1e6)
```

```
chunks: 16
100%|
                    | 1/1 [00:03<00:00,  3.58s/it]
26998 total zeros
```

[35]:
```
prod.shape
```

[35]:
```
(1043088,)
```

[36]:
```
from stat_tests import chi2_test, ks_test

u = np.log10(prod) % 1

# u = multi_floatarr_from_chunks(u, n_mixes=1, n=2)

# for i in range(10):
#     u = np.concatenate((u[1::2], u[::2]))

chi2_test(u), ks_test(u)
```

[36]:
```
((array([[13.25247343,  0.15151092]]), 23.209251158954356),
 (array([[5.98410311e-04, 8.48934877e-01]]), [0.001593492000171268]))
```

[37]:
```
rng_bits = get_bitstring(u)
rng_bits = "".join(rng_bits)

# rng_bits = binary_tree_walk(u).astype(str)
# rng_bits = "".join(rng_bits)
```

[38]:
```
len(rng_bits)
```

[38]: 23991024

[39]:
```
RNG_test(rng_bits[:10**6])
```

```
100%|
                  | 16/16 [00:07<00:00,  2.14it/s]
```

[39]:

	test	p
0	Frequency Test (Monobit)	0.81
1	Frequency Test within a Block	0.81
2	Run Test	0.63
3	Longest Run of Ones in a Block	0.67
4	Binary Matrix Rank Test	0.77
5	Discrete Fourier Transform (Spectral) Test	0.27
6	Non-Overlapping Template Matching Test	0.90
7	Overlapping Template Matching Test	0.95
8	Maurer's Universal Statistical test	0.19
9	Linear Complexity Test	0.29
10	Serial test	0.66
11	Approximate Entropy Test	0.97
12	Cummulative Sums (Forward) Test	0.28
13	Random Excursions Test	0.06
14	Random Excursions Variant Test	0.19

[ ]:

[ ]: