UR PSSM and Logo

Information Content and Sequence Logo Functions

The implementation below is based on the paper "Information Content of Binding Sites of Nucleotide Sequences".

We compute the information content (entropy) of each nucleotide position, i.e., how different they are from "random". For each nucleotide location along the motif $i = 1, \ldots, n$, let

$$R_i := H_g + \sum_{b=A}^{T} p_{b,i} \log 2(p_{b,i}),$$

denote the decrease in entropy from "random" ($H_g := 2$) at location i, where $p_{b,i}$ is the probability of base b at position i, which is estimated using the PPM entries (normalized nucleotide counts in each position). One can then compute the total decrease (total information gained) by $R = \sum_{i=1}^{n} R_i$. A sampling error correction can be applied to correct for biases that arises from using PPM and not the actual probabilities (see the Appendix in the paper).

In [9]:

```
from Bio import motifs, SeqIO
from Bio.Seq import Seq
import numpy as np
import math
import matplotlib as mpl
from matplotlib.text import TextPath
from matplotlib.patches import PathPatch
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt

def compute_pssm(lseq, pscount={'A':0.25,'C':0.25,'G':0.25,'T':0.25}, \
                 background={'A':0.25,'C':0.25,'G':0.25,'T':0.25}):
    '''Computes the PFM, PWM, and PSSM of a list of nucleotide sequences'''
    m = motifs.create(list(map(Seq, lseq)))

    pfm = m.counts
    # pwm with no pscount is a stochastic matrix
    pwm = m.counts.normalize(pseudocounts=pscount)
    pssm = pwm.log_odds(background)
    return pfm, pwm, pssm, m, m.consensus


# The following code is based on:
# https://github.com/saketkc/motif-logos-matplotlib/blob/master/Sequence%20log
os%20in%20Python.ipynb
# However, the code there has few errors. The code below follows the paper "In
formation Content
# of Binding Sites of Nucleotide Sequences"
```

```python
def calc_IC_approx_err(motif):

    '''Approximate calculate of small-sample correction error'''
    print('Computing approximate correction error...')
    bases = list(motif.pwm.keys())
    n = len(motif.counts[bases[0]])  # sequence length
    return (len(bases)-1)/(2 * n * np.log(2))

def calc_IC_exact_err(motif):
    '''Exact computation of small-sample correction error'''
    ##  O(n^3)
    print('Computing exact correction error...')
    pwm = motif.pwm
    bases = list(pwm.keys())
    n = na = len(motif.counts['A'])  # sequence length
    nc = ng = nt = exact_error = 0
    done = False
    while not done:
        #print (na,nc,ng,nt)
        pp = (0.25**na)*(0.25**nc)*(0.25**ng)*(0.25**nt)
        frac = pp*math.factorial(na+nc+ng+nt)/(math.factorial(na)*math.factori
al(nc)*\
                                               math.factorial(ng)*math.factori
al(nt))
        exact_error += frac*sum([-p*np.nan_to_num(np.log2(p)) for p in \
                                 [na/n, nc/n, ng/n, nt/n]])
        if nt<=0:
            ## iterate inner loop
            if ng > 0:
                ## g => t
                ng = ng - 1
                nt = nt + 1
            elif nc > 0:
                ## c -> g
                nc = nc - 1;
                ng = ng + 1;
            else:
                ## a->c
                na = na - 1
                nc = nc + 1
        else:
            if ng > 0:
                ## g => t
                ng = ng - 1
                nt = nt + 1
            elif nc>0:
                ## c => g; all t -> g
                nc = nc - 1
                ng = nt + 1
                nt = 0
            elif na>0:
                ## a => c; all g,t -> c
                nc = nt + 1
                na = na - 1
                nt = 0
            else:
                done = True
    return exact_error
```

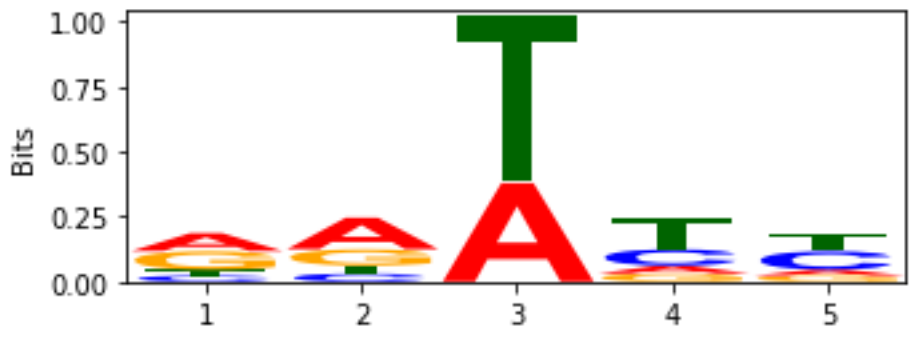
```
def calc_info_content(motif, corr_type = 'no'):
    '''Calculate information content with small sample correction.
    Note that for both corr_type=='approx' (should be used for
    sequences larger than 50 nt) and for corr_type=='exact' (should
    be used for sequences smaller than 50 nt), the output can attain
    both negative and positive values. See the paper "Information Content
    of Binding Sites of Nucleotide Sequences". Thus, for sequence logos, use
    the default (i.e. corr_type = 'no)
    '''
    pwm = motif.pwm  # should not use relative information
    bases = list(pwm.keys())
    if corr_type=='no':
        Hg = np.log2(len(bases))
    elif corr_type=='approx':
        Hg = np.log2(len(bases)) - calc_IC_approx_err(motif)
    else:  # exact
        Hg = calc_IC_exact_err(motif)
    #print('Hg = {}'.format(Hg))
    return [Hg+sum([pwm[b][l]*np.nan_to_num(np.log2(pwm[b][l])) for b in bases
])\
            for l in range(0, len(motif))]

def calc_rel_info(motif, corr_type = 'no'):
    '''Calculate relative information, i.e. the information content
    of each base along the sequence'''
    info_cont = calc_info_content(motif, corr_type)
    return {b: [np.nan_to_num(p*i) for p, i in zip(motif.pwm[b], info_cont)] \
            for b in list(pwm.keys())}

def gen_nt_sequence_logo(rel_info, fig_size=(5,2)):
    '''Generates nucleotide sequence logo.
    rel_info is computed by: rel_info=calc_rel_info(motif, 'no')'''
    fp = FontProperties(family="Arial", weight="bold")
    globscale = 1.35  # this, and the values here below were set for family="A
rial"
    LETTERS = { 'T' : TextPath((-0.305, 0), "T", size=1, prop=fp),
                'G' : TextPath((-0.384, 0), "G", size=1, prop=fp),
                'A' : TextPath((-0.35, 0), "A", size=1, prop=fp),
                'C' : TextPath((-0.366, 0), "C", size=1, prop=fp) }

    COLOR_SCHEME = {'G': 'orange',
                    'A': 'red',
                    'C': 'blue',
                    'T': 'darkgreen'}

    def letterAt(letter, x, y, yscale=1, ax=None):
        '''Plots a letter at a given position with a given scale.'''
        text = LETTERS[letter]

        t = mpl.transforms.Affine2D().scale(1*globscale, yscale*globscale) + \
            mpl.transforms.Affine2D().translate(x,y) + ax.transData
        p = PathPatch(text, lw=0, fc=COLOR_SCHEME[letter], transform=t)
        if ax != None: ax.add_artist(p)
        return p

    fig, ax = plt.subplots(figsize=fig_size)
    fig, ax = plt.subplots(figsize=fig_size)
    x = 1

    maxi = 0
    for i in range(0, len(list(rel_info.values())[0])):
        scores = [(b,rel_info[b][i]) for b in COLOR_SCHEME.keys()]
        scores.sort(key=lambda t: t[1])
        y = 0
        for base, score in scores:
            letterAt(base, x, y, score, ax)
            y += score
        x += 1
        maxi = max(maxi, y)
    plt.xticks(range(1,x))
    plt.xlim((0, x))
    plt.ylim((0, maxi))

    return fig, ax
```

An Example - UR PSSM

In [23]:

```
import pandas as pd
import os
import numpy as np
from Bio import motifs, SeqIO
from Bio.Seq import Seq
import re
from collections import deque
from termcolor import colored
from pprint import pprint

mlen = 5
spec = 'dsDNA_Bacteria'
top_num = 12 #9
lfile = 'seq_lfile.png'
# ==============================================================================
============
dtype_rnd = {'signf_sorted':str, 'tot_num_Fall_sort':int}
UR_rnd_xlsx_file = '/Users/yoramzarai/work/school/Simulation/Viruses/Data_stat
s/UR_rnd_m'\
+str(mlen)+'_'+spec+'.xlsx'

df_rnd = pd.read_excel(UR_rnd_xlsx_file, header=0, dtype=dtype_rnd)

columns_names = list(df_rnd.columns)
# print('Found {} columns:'.format(len(columns_names)))
# for i, s in enumerate(columns_names, 1): print(i,s,sep='. ')
top_UR = {df_rnd.loc[s][columns_names[0]] : df_rnd.loc[s][columns_names[1]] fo
r s in range(top_num)}
eql_nt = {s for s in top_UR.keys() if len(set(s))==1}
diff_nt = set(top_UR.keys()) - eql_nt
print('eql:', eql_nt)
print('diff:', diff_nt )

# compute motif count matrices
backg={'A':0.25,'C':0.25,'G':0.25,'T':0.25}
pfm, pwm, pssm, m, cons = compute_pssm(diff_nt, backg, backg)
#print(pfm, pwm, pssm, sep='\n')

# generate sequence logo
rel_info = calc_rel_info(m, 'no')
fig, ax = gen_nt_sequence_logo(rel_info)
ax.set_ylabel('Bits')
plt.axis([0.5, mlen+0.5, 0, ax.get_ylim()[1]])
plt.tight_layout()
if lfile!='':
    plt.savefig(lfile, dpi=200)
    print('Figure saved in {}'.format(lfile))
```

```
eql: {'AAAAA', 'CCCCC', 'TTTTT', 'GGGGG'}
diff: {'AATTT', 'GATCA', 'AGATC', 'TTTTC', 'GATCT', 'GGATC', 'AAAA
T', 'CCTGG'}
Figure saved in seq_lfile.png

/opt/local/Library/Frameworks/Python.framework/Versions/3.7/lib/py
thon3.7/site-packages/ipykernel_launcher.py:102: RuntimeWarning: d
ivide by zero encountered in log2
```


Bits