Pattern_match_regex

May 13, 2019

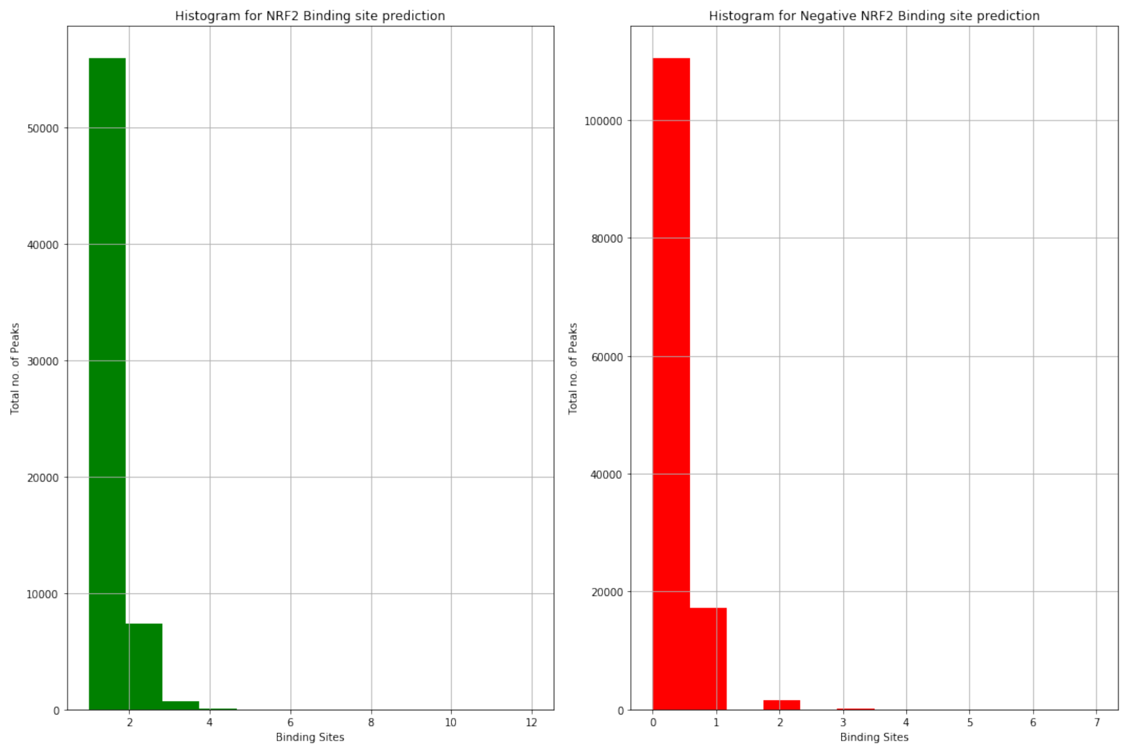
[2]:
```
#Finding Putative NRF2 Binding Sites Using Motifs and then Visualize them
import time
import gzip
import shutil
import pandas as pd
import numpy as np
import re
from Bio import SeqIO
from itertools import islice
import matplotlib.pyplot as plt
import twobitreader as tbr
```

[3]:
```
#Creating Reverese Complements
def reverseComp(Seq):
    seq = Seq.upper()
    d = {'A':'T', 'T':'A', 'G':'C', 'C':'G'}
    try:
        seq = seq[::-1]
        rc_seq = "".join([d[nuc] for nuc in seq])
    except KeyError:
        return "Not Viable DNA Seq"
    return rc_seq
```

[14]:
```
motif = '[AGC]TGA[CTG][ATCG][GCAT][AGT]GC[ATCG]'
regBS = re.compile(motif)
motifDF = []
motifQuant = []
genome = tbr.TwoBitFile('/Users/kalyanidhusia/Downloads/hg19.2bit')
with open('/Users/kalyanidhusia/Desktop/nrf2_R/ENCFF126HBJ.bed') as f:
    for line in f:
        if line.startswith('track') == False:
            peak = list(line.split())
            seq = (genome[peak[0]][int(peak[1]):int(peak[2])]).upper()
            rSeq = reverseComp(seq)
            sequences = []
            sequences.extend(re.findall(regBS, seq))
            sequences.extend(re.findall(regBS, rSeq))
            if len(sequences) > 0:
                seqs = pd.DataFrame({'binding':sequences, 'chrom':peak[0],␣
 ↪'chromstart':peak[1], 'chromend':peak[2], 'NR':'NRF2'})
                motifDF.append(seqs)
                motifQuant.append([peak[0], peak[1], peak[2], len(seqs),␣
 ↪len(seq)])
search_reg = pd.concat(motifDF)
names = ['chrom', 'chromstart', 'chromend', 'numOfMatches', 'lenSeq']
dist_reg = pd.DataFrame(motifQuant, columns=names)
dist_reg.head()
n = 5
x = [len(i[6+n:-6-n]) for i in search_reg['binding']]
```

[36]:
```
motif = '[AGC]TGA[CTG][ATCG][GCAT][AGT]GC[ATCG]'
regBS = re.compile(motif)
motifDF = []
motifQuant = []
genome = tbr.TwoBitFile('/Users/kalyanidhusia/Downloads/hg19.2bit')
with open('/Users/kalyanidhusia/Desktop/nrf2_R/search2/min_score70/DNAse/
 ↪negativenrf2/sudin_negative_nrf2.bed') as f:
    for line in f:
        if line.startswith('track') == False:
            peak = list(line.split())
            seq = (genome[peak[0]][int(peak[1]):int(peak[2])]).upper()
            rSeq = reverseComp(seq)
            sequences = []
            sequences.extend(re.findall(regBS, seq))
            sequences.extend(re.findall(regBS, rSeq))
            if len(sequences) >= 0:
                seqs = pd.DataFrame({'binding':sequences,'chrom':
 ↪peak[0],'chromstart':peak[1],'chromend':peak[2],'NR':'NRF2'})
                motifDF.append(seqs)
                motifQuant.append([peak[0], peak[1], peak[2], len(seqs),␣
 ↪len(seq)])
zsearch_reg = pd.concat(motifDF)
names = ['chrom', 'chromstart', 'chromend', 'numOfMatches', 'lenSeq']
zdist_reg = pd.DataFrame(motifQuant, columns=names)
dist_reg.head()
n = 5
x = [len(i[6+n:-6-n]) for i in zsearch_reg['binding']]
```

[44]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
plt.subplot(121)
plt.title('Histogram for NRF2 Binding site prediction')
plt.xlabel('Binding Sites')
plt.ylabel('Total no. of Peaks')
plt.grid(True)
plt.hist(dist_reg['numOfMatches'], bins= 12, color ="green")
plt.subplot(122)
plt.title('Histogram for Negative NRF2 Binding site prediction')
plt.xlabel('Binding Sites')
plt.ylabel('Total no. of Peaks')
plt.grid(True)
plt.hist(zdist_reg['numOfMatches'], bins= 12, color ="red")
plt.tight_layout()
plt.savefig('compare_hist')
```



[7]:
```
#Creating the LOGO for NRF2 Motif using weblogo
from Bio.Seq import Seq
from Bio import motifs
instances = search_reg['binding']
m = motifs.create(instances)
m.weblogo("logo_pos_outzero.tiff")
```

[47]:
```
cluster1 = search_reg['binding']
cluster2 = zsearch_reg['binding']

clusters = [cluster1, cluster2]
for i in range(len(clusters)):
    instances = []
    for seq in clusters[i]:
        instances.append(Seq(seq[:6+n] + seq[-6-n:]))
    m = motifs.create(instances)
    m.weblogo('motifs' + str(i) + '.png')
```

[48]:
```
#One Hot Encoding Sequence
def oheSeq(DNAString):
    seq = DNAString.upper()
    nuc = 'ATGC'
    char2int = dict((c, i) for i, c in enumerate(nuc))
    int2char = dict((i, c) for i, c in enumerate(nuc))
    integer_encoded = [char2int[char] for char in seq]
    OHE = []
    for value in integer_encoded:
        letter = [0 for _ in range(len(nuc))]
        letter[value] = 1
        OHE.append(letter)
    seq_ohe = np.asarray(OHE)
    return seq_ohe
```

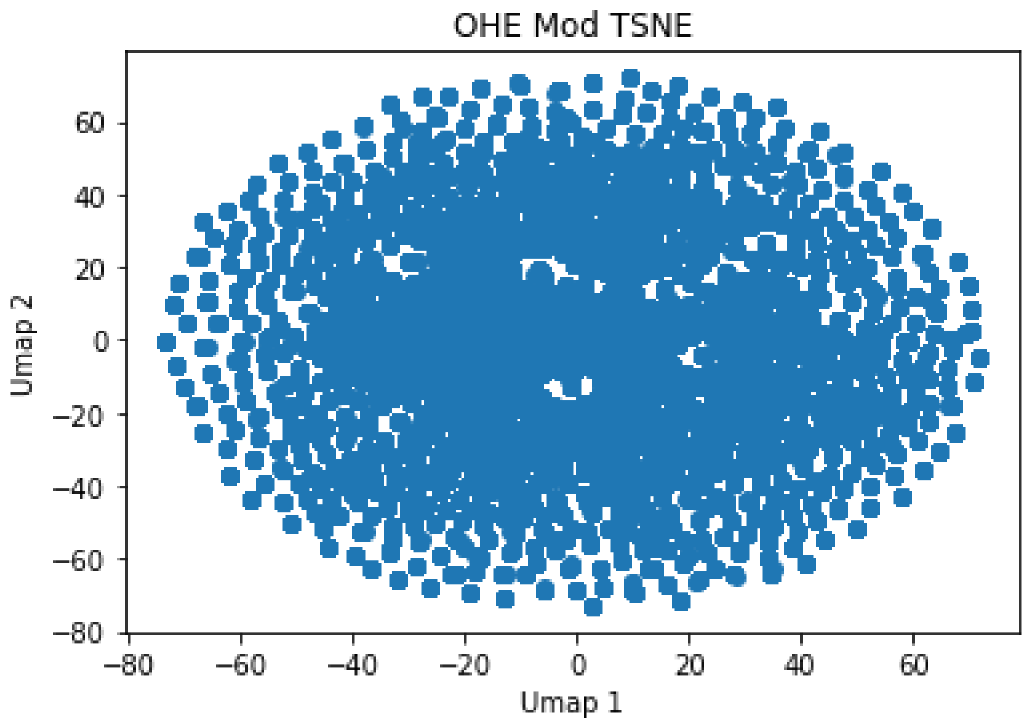
[52]:
```
#modified OHE for interspaced regions
def oheSeqMod(DNAString, flank_len):
    seq = DNAString.upper()
    flanks = seq[:6+flank_len] + seq[-6-flank_len:]
    nuc = 'ATGC'
    char2int = dict((c, i) for i, c in enumerate(nuc))
    int2char = dict((i, c) for i, c in enumerate(nuc))
    integer_encoded = [char2int[char] for char in flanks]
    OHE = []
    for value in integer_encoded:
        letter = [0 for _ in range(len(nuc))]
        letter[value] = 1
        OHE.extend(letter)
    OHE.append(len(seq[6+flank_len:-6-flank_len]))
    return OHE
```

[53]:
```
#Dimensionality Reductions
#euc for euclidean
euc_ohe =  np.array([oheSeqMod(i, 5) for i in search_reg['binding']])
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import umap
#Be careful with umap installation pip install umap will cause you to install␣
 ↪the wrong ver of umap
#(and will also break the real module)

pca = PCA(n_components=2)
pca.fit(euc_ohe)
```

```
print(pca.explained_variance_ratio_)

#Takes a bit with larger datasets (scales n^2 in both compute time and memory)
X_embedded = TSNE(n_components=2).fit_transform(euc_ohe)

#Also takes a bit (even though they claim it is faster than TSNE)
umapped = umap.UMAP().fit_transform(euc_ohe)
```

```
[0.13439471 0.09280595]

/Users/kalyanidhusia/anaconda3/lib/python3.6/site-packages/umap/spectral.py:229:
UserWarning:

Embedding a total of 57 separate connected components using meta-embedding
(experimental)
```

[54]:
```
#Graphing example
dim1 = X_embedded[:, 0]
dim2 = X_embedded[:, 1]
plt.scatter(dim1, dim2)
plt.title("OHE Mod TSNE")
plt.xlabel('Umap 1')
plt.ylabel('Umap 2')
plt.show()
```



[131]:
```
instances = search_reg['binding']
instances.head()
```

[131]:
```
0    ATGACTCAGCA
0    ATGACGGAGCA
0    ATGACTCAGCA
1    ATGAGTGGGCT
0    GTGACTCAGCG
Name: binding, dtype: object
```

[55]:
```
%matplotlib notebook

import numpy as np
import pylab as plt
import seaborn as sns; sns.set()

# change the path!
import sys; sys.path.append('/Users/kalyanidhusia/Downloads/FIt-SNE-master/')
from fast_tsne import fast_tsne
```

[ ]:
```
(x_train, y_train), (x_test, y_test) = euc_ohe.load_data()
x_train = x_train.reshape(60000, 784).astype('float64') / 255
x_test  =  x_test.reshape(10000, 784).astype('float64') / 255
X = np.concatenate((x_train, x_test))
y = np.concatenate((y_train, y_test))
print(X.shape)

# Do PCA and keep 50 dimensions
X = X - X.mean(axis=0)
U, s, V = np.linalg.svd(X, full_matrices=False)
X50 = np.dot(U, np.diag(s))[:,:50]

# We will use PCA initialization later on
PCAinit = X50[:,:2] / np.std(X50[:,0]) * 0.0001

# 10 nice colors
col = np.array(['#a6cee3','#1f78b4','#b2df8a','#33a02c','#fb9a99',
                '#e31a1c','#fdbf6f','#ff7f00','#cab2d6','#6a3d9a'])
```

[1]:
```
!export PATH=/Library/TeX/texbin:$/Users/kalyanidhusia/anaconda3/bin:/Users/
↪kalyanidhusia/anaconda3/bin:/Users/kalyanidhusia/anaconda3/condabin:/usr/bin:
↪/bin:/usr/sbin:/sbin
```

[3]:
```
jupyter nbconvert --Pattern_match_regex.ipynb --to pdf
```

```
  File "<ipython-input-3-9fcb77b5e400>", line 1
jupyter nbconvert --Pattern_match_regex.ipynb --to pdf
```

```
                        ^
SyntaxError: invalid syntax
```

[ ]:

```

```